ARL-TR-7683 ● May 2016

US Army Research Laboratory

A Guide for Developing Human-Robot Interaction Experiments in the Robotic Interactive Visualization and Experimentation Technology (RIVET) Simulation

by Kristin E Schaefer and Ralph Brewer

ARL-TR-7683 ● MAY 2016

US Army Research Laboratory

A Guide for Developing Human-Robot Interaction Experiments in the Robotic Interactive Visualization and Experimentation Technology (RIVET) Simulation

by Kristin E Schaefer
Oak Ridge Associated Universities, Oak Ridge, TN

Ralph Brewer
Vehicle Technology Directorate, ARL

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* May 2016	**2. REPORT TYPE** Final	**3. DATES COVERED (From - To)** September 2013–December 2015
4. TITLE AND SUBTITLE A Guide for Developing Human-Robot Interaction Experiments in the Robotic Interactive Visualization and Experimentation Technology (RIVET) Simulation		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Kristin E Schaefer and Ralph Brewer		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Research Laboratory ATTN: RDRL-HRS-E Aberdeen Proving Ground, MD 21005-5425		**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-7683
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The Robotic Interactive Visualization and Experimentation Technology (RIVET) is a computer-based simulation system that was developed to merge game-based technologies with current and next-generation robotic development. The original design of RIVET specifically addressed engineering-related functionality, including the capability to test critical algorithms prior to field testing a robotic system, perform rapid consecutive test scenarios to find software bugs, and conduct algorithm verification across a wide spectrum of test scenarios. While these functional test procedures have been shown to be essential, the design of this game-based platform also lends itself to human-in-the-loop (HITL) experimentation. This technical report outlines the capabilities of RIVET as a Human Factors platform for HITL human-robot interaction (HRI) experimentation. It further provides a primer for HRI experimentation development for the nonprogrammer with example TorqueScript files and step-by-step instructions to develop the virtual environment. Additional recommendations for collecting subjective and objective human data related to HRI are discussed.

15. SUBJECT TERMS
human factors, human-in-the-loop, human-robot interaction, RIVET, simulation

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Kristin E Schaefer
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	162	**19b. TELEPHONE NUMBER (Include area code)** 410-278-5972

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

Acknowledgments

This research was supported in part by an appointment to the US Army Research Postdoctoral Fellowship Program administered by the Oak Ridge Associated Universities through a cooperative agreement with the US Army Research Laboratory (ARL). Research was sponsored by ARL and was accomplished under Cooperative Agreement Number W911-NF-12-2-0019.

We would like to acknowledge Dave Wagner, formerly with General Dynamic Robotic Systems, for his work on the development of the Robotic Interactive Visualization and Experimentation Technology (RIVET) platform and technical support. We would also like to thank Eric Avery of ARL's Human Research and Engineering Directorate (HRED) for his assistance in adapting the CARVE software, and Tom Smith (HRED) for his support.

INTENTIONALLY LEFT BLANK.

1. Summary

This technical report outlines the capabilities of Robotic Interactive Visualization Experimentation Technology (RIVET), a computer-based simulation environment, as a Human Factors platform for human-in-the-loop (HITL) human-robot interaction (HRI) experimentation. This system was originally designed by the US Army Research Laboratory's (ARL) Robotics Collaborative Technology Alliance (ARL 2012). The purpose for the simulation tool was to provide engineers with a means to test and debug intelligence and perception algorithms for autonomous unmanned vehicles prior to field exercises. Here, we have adapted the software to allow users to work cooperatively with unmanned systems to address the human element of HRI. HRI experimentation using simulation supports the capability to assess performance, individual differences, preferences, concerns, and potential issues that may directly or indirectly impact the design of the system for multiple future Soldier-robot teaming operations. This report provides a primer for the nonprogrammer with example TorqueScript files and step-by-step instructions to develop virtual environments (VEs) for HRI experimentation. Additional recommendations for subjective and objective human data collection tools related to HRI are discussed.

2. Introduction

ARL's Intelligent Systems Enterprise vision is to enable the teaming of autonomous intelligent systems with Soldiers in dynamic, unstructured combat environments, as well as in noncombat military installations and base operations. To accomplish this vision for interdependent Soldier-robot teaming, there has been a paradigm shift in robotic research conducted by ARL from the current instantiation of fielded remote-controlled or teleoperated robots to systems with increased intelligence, decision-making capability, and autonomy (Groom 2008; Chen and Terrence 2009; Phillips et al. 2011; Schaefer 2013). This type of teaming is needed for future joint, interdependent, network-enabled operations.

While the technological capabilities of robotic systems are advancing by the day, many of these new systems are still in the early stages of research and development. In many cases, areas of need or potential use have been identified, preliminary requirements have been created, and engineering solutions for prototype systems have been researched. However, the human element must be considered early on in this design process, because without considering human factors, such as human-system interfaces, performance, as well as trust and expectation, the potential result will lead to limited or inappropriate use of the system (Parasuraman and Riley 1997; Lee and See 2004).

Computer-based simulation provides a safe, economical, and efficient means to assess HRI throughout the robot life cycle. The following sections of the introduction include a brief overview of HRI and HRI metrics, a description of how HRI can be included into robotic technology research and development, and an overview of the benefits of using simulation for HRI experimentation. This will then lead to a description and main purpose of this report: how to use the RIVET computer-based simulation system for HRI experimentation. A primer for the nonprogrammer with example TorqueScript files and step-by-step instructions to develop the VE is provided in the appendixes.

2.1 Human-Robot Interaction

HRI is "a field of study dedicated to understanding, designing, and evaluating robotic systems for use by or with humans" (Goodrich and Schultz 2007). The study of HRI is a relatively new field (emerging in the mid-1990s), but one that is gaining traction because of the increase of robotic technology in both the public and government, particularly defense applications. As with any new field of study, researchers are still developing a common understanding of metrics and evaluation techniques. Many different researchers have attempted to develop a common ground for understanding and measuring HRI, each of which has provided some additional insight into this problem. However, specific lines of research have taken specific perspectives to experimentation as a means to obtain generalizable knowledge. A problem with this type of methodology is that it results in context-specific or task-specific findings that may or may not be generalizable to other domains or applications. In addition, a number of measurement techniques exist, including self-report measures, behavioral measures, task performance, and psychophysiological measures (Bethel et al. 2007). Each type of measurement has its own unique benefits and limitations. In the following paragraphs, we briefly reference some key findings that may be helpful as researchers seek to better incorporate HRI assessment into computer-based simulation research.

We begin by highlighting the work by Steinfeld et al. (2006), which attempts to identify metrics that can be used across robot domains and applications. They suggest the importance of clearly identifying what they call the "task-oriented" requirements, instead of the "task-specific" requirements. This publication is a well-recognized description of common metrics for HRI. The common task-oriented metrics identified in this work include navigation, perception, management, manipulation, and social interaction. The following bullets list the associated measureable items for each metric.

- *Navigation* throughout the environment: effectiveness (percent of tasks complete, deviation from planned route, and number of obstacles avoided); efficiency (time to complete task, operator time for task, and average time for obstacle extraction); and workload (the number of interventions by the user);
- *Perception* about objects in the environment based on sensor input: detection measures; recognition measures, such as classification; judgments about the distance, size, or length of the environment; estimates of absolute and relative motion;
- *Management* of the actions of humans and robots: fan out; intervention response time; level of autonomy discrepancies;
- *Manipulation* of the robot interacting with the environment, such as grasping, pushing, or payload drop-off: degree of mental computations and contact errors; and
- *Social interaction*: interaction characteristics; persuasiveness of the robot; trust; engagement; compliance.

Goodrich and Schultz (2007) identified key challenges specific to uncertainty related to unstructured and extreme environments; risk to people due to their proximity and vulnerability in the interaction; the integration of appropriate social and emotional aspects of interaction; and issues with integrating natural language into human-robot communication paradigms. They identified multiple design-based attributes that can affect the interaction of humans with robots. These included the following:

- The level and behavior of autonomy: neglect tolerance;
- Nature of information exchanged: interaction time, mental workload, situation awareness, common ground;
- Structure of the team: fan-out, role of the team;
- Adaptation, learning, and training of people and the robot; and
- Task-shaping: goal-directed task analyses, cognitive work analyses, and ethnographic studies.

More recently, Weiss et al. (2009) took a social approach and developed an evaluation framework for measuring HRI. This framework focuses on the usability (e.g., effectiveness, efficiency, learnability), social acceptance (e.g., performance expectations, effort expectancy, attitudes toward the technology), user experience, and societal impact (e.g., consequences to the social life of a specific community following the introduction of a robot).

As robotic design continues to move toward interactive collaborative teaming with people, additional HRI metrics have emerged. These first specify the importance of communication, highlighting specific elements that can be measured from the human. For example, analyzing speech and gesture patterns can be used to identify goal inference, anticipatory action selection, and congruence of information (Bicho et al. 2010). This type of analysis may also benefit from identifying speech-based interaction in terms of both action-only speech versus action speech with a descriptor. Cassenti et al. (2011) found that people were more successful in directing robots during a navigation task if location labels (e.g., structural parts of a building) were used and understood by the robot. A second recent area of study associated with HRI metrics has identified the importance of measuring situation awareness. This can be accomplished by assessing the stages of information processing (i.e., information acquisition, decision making, and action implementation), control allocation, attentional control, multitasking, and task switching (Chen and Barnes 2014).

Most recently, Murphy and Schreckenghost (2013) reviewed 29 papers regarding metrics associated with HRI. They reported that this is an area still being developed and understood. Their key findings related to the human-robot team or system are reported as follows:

- Productivity (effectiveness): team productivity, task difficulty, and time (autonomous versus manual operations)
- Efficiency: amount of effort, human-robot ratio, interaction, ratio of operator time to robot time, and total time to complete task
- Reliability: false alarms, flexibility, interventions, level of automation discrepancies, similarity, and time (intervention response time, unscheduled manual operations)
- Safety: risk to people, robot awareness of people
- Coactivity: cognitive interaction, crypsis coefficient, degree of monotonicity, neglect tolerance, plan state, percent of requests from operator and robot, task allocation, and time in unscheduled manual operations

As can be seen by this brief review of the HRI domain, there is a direct interplay between use and design that should be further explored and understood. In the following subsection we begin to address the importance of including HRI into simulation.

2.2 HRI and Computer-Based Simulation

According to Dautenhahn (2007), incorporating HRI evaluation into the robot's design and development process has a number of challenges that stem from integrating an interdisciplinary approach with technological design. However, taking the human into consideration throughout this process may help to alleviate previously established issues related to nonuse, misuse, disuse, or abuse of developed systems (Parasuraman and Riley 1997), leading to a greater return on investment. Additionally, HITL research and development becomes progressively more important as robots continue to move into an integrated team member role.

Simulation provides the opportunity for collaboration between Soldiers and unmanned platforms at multiple levels in the design process. This can help to guide the design process and provide needed information back into development. The inclusion of simulation into the design process is not a new concept. For example, simulation is a cost-effective strategy often used throughout the entire life cycle of software development to identify real-time performance shortfalls and to develop risk-resolution techniques (Boehm 1988). Here we suggest that the modeling and simulation process is important for the study of human and robot interaction. Loper (2015) describes the modeling and simulation process in the following steps:

- Establish purpose and scope: problem statement
- Formulate a conceptual model: abstraction of the real-world system under investigation, goals
- Acquire and analyze data: requirements gathering
- Develop simulation model and program: operational model and computer implementation
- Verify and validate model and simulation: accuracy of conceptual model, requirements, and representation of the real-world
- Design experiments with specific details: length, number of runs/replications, manner of initialization (e.g., training)
- Execute simulation and analyze output: estimate measures of performance, and update analysis and design elements

For the purposes of this work, we are focusing on virtual or computer-based simulation. More specifically, we are interested in the benefits of HITL simulation at the research and development stages. While one key benefit of simulation is rapid and systematic development and assessment of technology, the process by which a human uses or understands the benefits of the technology can directly impact the process by which a system will be used in the future. Here, we suggest that incorporating HITL simulation experimentation early on in the research and development process can benefit the analysis, design, and development stages for robotic technology.

Computer-based simulation has become an important part of HITL experimentation, and provides a number of benefits, primarily those related to cost and safety across a number of applications areas. More recently, this type of experimentation has been used to assess HRI. Types and uses for simulation include prototype design and development (e.g., algorithm testing, sensor design, and control interfaces), training and practice interacting with robotic systems, and transitions to real-world robotics.

Within HRI, benefits of simulation include time and cost benefits due to rapid and repeatable assessment of HRI. The recommendation for simulation use for HRI experimentation took root in the 1980s with Sheridan’s (1986) research into the human’s supervisory control of robotic space systems. While he recommended that both computer simulation and hardware prototype development are needed for design and development, a key benefit of simulation is *time* due to the quick and systematic adaptations of the robotic system. In the following decade, research demonstrated cost-related benefits through a reduction in development time and required financial need. For example, Michel (1998) demonstrated this cost benefit through the development of the Webots simulator for robotic prototype development. This simulator allowed for the successful design of the mechanical structure of robots, as well as the development of intelligent controllers. The success of this approach was attributable to the rapid and repeated testing that was not possible with real-world systems. In addition, the scarcity and expense of real-world robotic systems also made simulation a viable option to test human perception, situation awareness, and teamwork, as seen with unmanned search and rescue robots (Nourbakhsh et al. 2005; Lewis et al. 2007).

The medical robotics community has identified some *training* benefits of integrating HRI principles into simulation systems. In a review of the research, Kunkler (2006) suggested that the similarities between computer simulation tools and robotic surgery systems (e.g., mechanized feedback, monitors to visualize task, and similar computer software for HRI) provided successful training opportunities for physicians, while maintaining personal and patient safety. In addition, Basdogan et al. (2004) suggested that the inclusion of haptic interfaces, rendering, and recording, as well as playback options into the computer simulation, can significantly improve training transfer to real-world robotic systems.

This goal of successful *transfer* from simulation to real-world HRI is a common theme across all domains of robotics. One common practice is the development of compatible source codes (e.g., robot motion [Davies 2000] and Webots [Michel 1998]). For example, in their research comparing a virtual and real-world Pioneer robot, Gerkey et al. (2003) suggested that while there is no guarantee for direct comparability between simulation and real-world robotic systems, there is promise of such transfer when testing new sensors or control interfaces prior to hardware development.

3. RIVET and CARVE for HRI Experimentation

Simulation has opened the door to understand and assess the human, as well as components of the interaction, early on in the design process. This allows researchers and roboticists to begin to assess the interaction-specific design elements proactively rather than retroactively. Many HRI studies have begun to rely on simulation for some element of experimentation. Many platforms are used to assess interaction with algorithms, such as planning algorithms (Sycara and Lewis 2003) and for testing human preferences and behaviors toward sensors (Hughes and Lewis 2004). Simulation is also used to study elements of communication with robots, including haptics (Her and Hsu 2001), semantic maps (Nielsen et al. 2004), and bidirectional communication (Rickheit and Wachsmuth 2008). Individual differences and elements of teaming have also been studied using simulation. These include topics such as workload and situation awareness (Parasuraman et al. 2003), as well as control of multiple robots (Olsen and Wood 2004; Chen et al. 2010). HRI-oriented simulation should also "accurately [reflect] the range of available information, behavior, and user experience encountered in actual robot operation" (Lewis et al. 2007, 101).

3.1 Overview of RIVET

In this section, we provide an overview of the Robotic Interactive Visualization and Experimentation Technology (RIVET) simulation environment. RIVET version 1.0 was built using the Torque Game Engine (TGE) version 1.5.2, designed by GarageGames. It was designed by the ARL's Robotics Collaborative Technology Alliance (RCTA) to allow engineers to test and debug intelligence and perception algorithms for autonomous unmanned vehicles prior to field exercises (ARL 2012). While field exercises provide a valuable insight on the performance of mobility and perception algorithms, the amount of time needed for field exercises must be used wisely. The cost to conduct data collection and testing in the field is substantially more than time spent in the lab conducting testing. This cost not only includes a financial cost, but also a time-associated cost in that it is difficult to organize and coordinate multidisciplinary integration sessions. Simulation tools like RIVET can provide an essential initial evaluation of robotic algorithms and concepts in simulated environments that can be varied systematically.

While RIVET was originally designed to test and debug algorithms, the design and capabilities of the software lend themselves to the development of HRI-specific missions with unmanned ground vehicles (UGVs), which is the focus of the remainder of this report. RIVET provides the underpinning for autonomous movement that allows experimentation regarding the interaction of Soldiers and these highly technical mobile vehicles. The following sections describe the extensive effort allocated to the adaptation and advancement of RIVET to be used for HRI experimentation purposes.

3.1.1 General Capabilities

Like most commercial off-the-shelf game engines, the TGE provides functionality for graphics, physics, artificial intelligence, lighting, and many other features. It provides all the necessary attributes and core functionality to conduct virtual experiments. Building a simulated environment involves several different activities necessary for HRI experimentation: 1) creating the virtual terrain surface, 2) adding static features typically found in the scene, 3) adding dynamic elements, such as people and vehicles, 4) producing the interactive graphical user interfaces (GUIs), 5) choosing the unmanned platform for the exercise, 6) implementing a user interface for the UGV, and finally, 7) assessing the user during a study. Specifics relating to each of these will be discussed in detail in Section 4 (Procedures for Using RIVET and HRI Experimentation).

3.1.2 Hardware/Software Considerations

The game engine adopts a client-server architecture. This is because it is designed for networking multiple systems together to play a game between multiple remote-located users. However, RIVET can be used in single-user mode or multi-user mode depending on the problem statement and research goals. In single-user mode, RIVET loads the scene and then a user is able to access the world editor, GUI editor, or control an avatar (Soldier, vehicle, etc.) through the mission area using keyboard or joystick. Multi-user mode allows up to 64 local area networked users to join the mission as clients. These clients can be another vehicle, robot, Soldier, or a sensor that is attached to an entity in the simulation. Currently, the choice of additional vehicle sensors within RIVET are cameras, laser detection and ranging (LADAR), and a custom interface used to connect other applications to the simulation environment. Sensors can be local to the initial server client machine or they can be a remote sensor as a client on a different machine. The key to determining which is right for the mission depends on the angle of view for the sensor. The local sensor view is what the server camera is projecting. If a different camera sensor view is needed, a separate computer running as a client will be used to connect to the vehicle, and then that graphics processor unit is used to develop the sensor scene (e.g., rear-view camera for traveling in reverse). General specifications relating to the quality of the computers, graphics cards, etc. can be found in the *RIVET Developer's Guide* (General Dynamics Robotics Systems 2010, 17).

3.1.3 RIVET Menu Options

Through the RIVET main menu (Fig. 1), there are customization options that are available through buttons at the bottom of the menu screen. It is possible to load a customized user mission (Server computer), join a mission (Client computer), or mount a sensor to a vehicle or robot on the RIVET server computer (Client computer). In addition, the configuration menu allows customization of the video screen size, resolution, and vehicles. Appendix A provides more RIVET menu options.

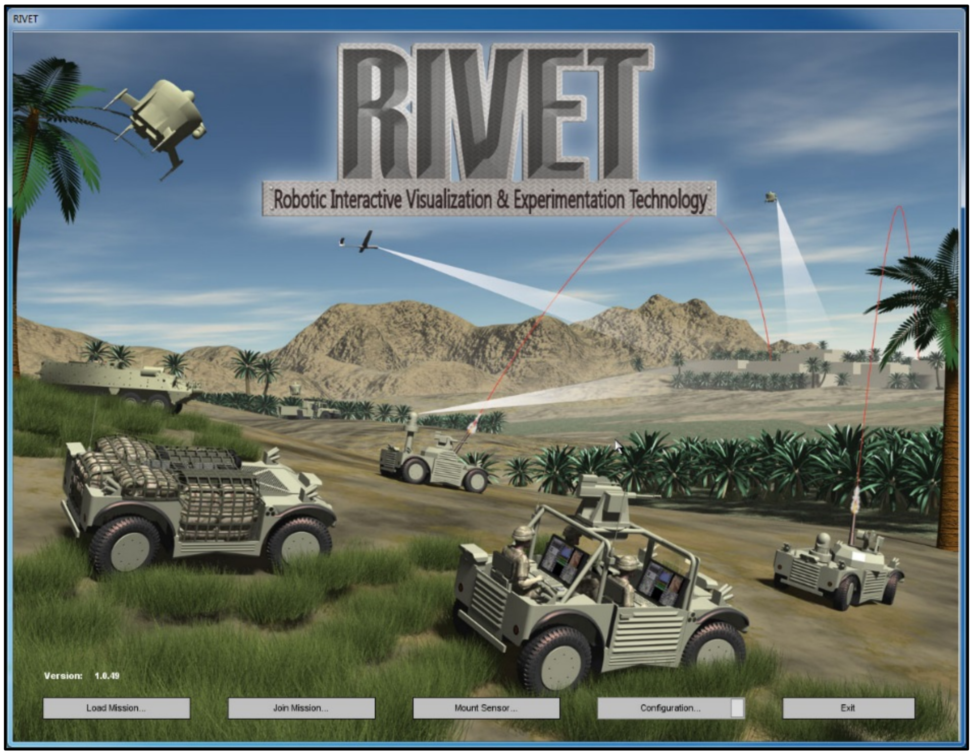


Fig. 1 RIVET Main Menu GUI

For a majority of our HRI-related research, we need to have more human interaction with the robot/autonomous vehicle than observing a camera sensor only. Therefore, the recommended setup is the 3-computer setup with the RIVET server, the additional client sensor, and the user interface application for extra control and integration of the UGV. The RIVET architecture also allows for the connection of outside applications by mounting a Basic Operations Layer Transmissions (BOLT) sensor to the vehicle. The BOLT libraries allow external interfaces to connect with the simulation (more information can be found in the Application Program Interface, which is included with the software). One such application is the Control for Autonomous Robotic Vehicle Experiments (CARVE) user interface application (discussed in detail in Section 3.2). The Server, Client, and CARVE Application computers are connected through an Ethernet local area network (LAN), using a Gigabit switch (Fig. 2).

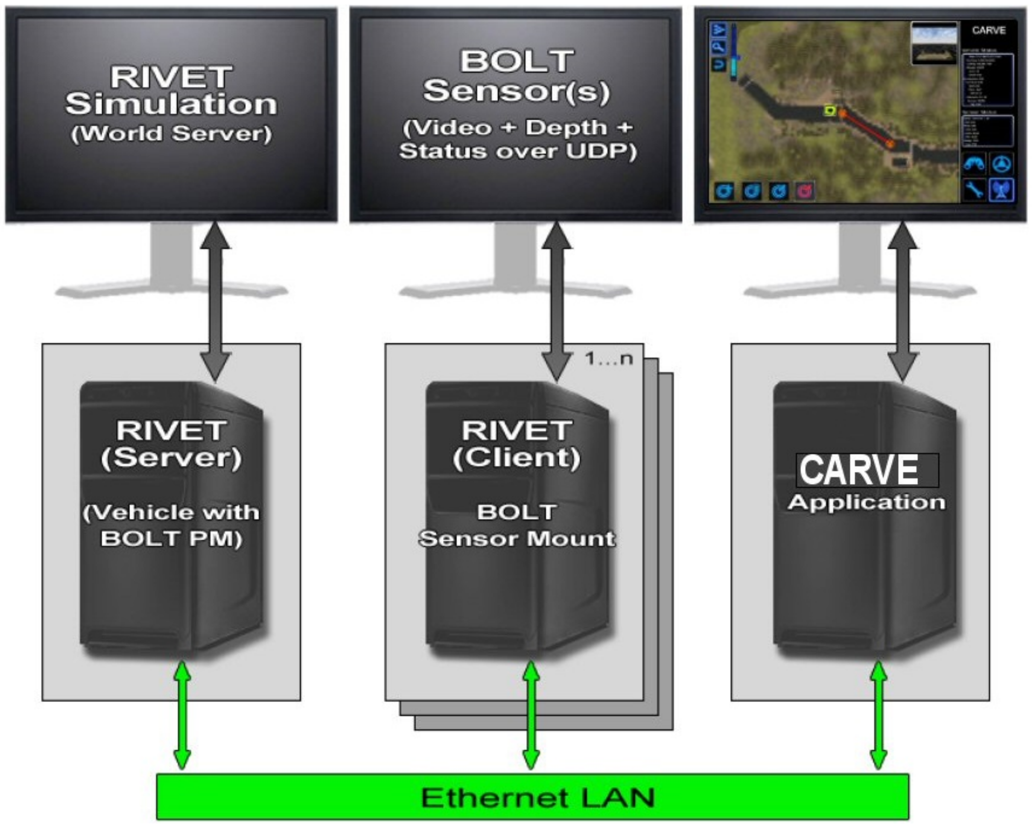

RIVET Simulation (World Server)
BOLT Sensor(s) (Video + Depth + Status over UDP)
CARVE
RIVET (Server) (Vehicle with BOLT PM)
1…n
RIVET (Client) BOLT Sensor Mount
CARVE Application
Ethernet LAN

Fig. 2 **RIVET and CARVE setup for single user, remote sensor, or GUI combination**

3.2 Overview of CARVE

CARVE is written in C Sharp (C#), allowing the use of all of the Microsoft .NET libraries. As its name implies, additional robotic vehicle control is the main functionality of CARVE. For example, CARVE provides customizable integration of new user interface controls for the simulated vehicle, as well as feedback displays to the participant. There was additional work completed to utilize CARVE as an autonomous driving station for human use experiments. This included the capability to mirror autonomous vehicle behavior through the integration of dynamic waypoints (in CARVE) and trigger points (in RIVET). Both RIVET and CARVE are customizable in terms of recording data from the vehicle (speed, health, time, location, etc.), environment (non-player character movement, triggers, and events), as well as user input (function allocation, time, number of interventions). Finally, CARVE provides additional capabilities and customizations to the RIVET platform to advance HRI experimentation, including visual feedback of vehicle statistics, dynamic map capabilities, and the inclusion of additional tasks (e.g., radar detection task). This added functionality and control is depicted in Fig. 3 and allows HRI assessment to occur early in development. Examples of the CARVE menus are provided in Appendix B.

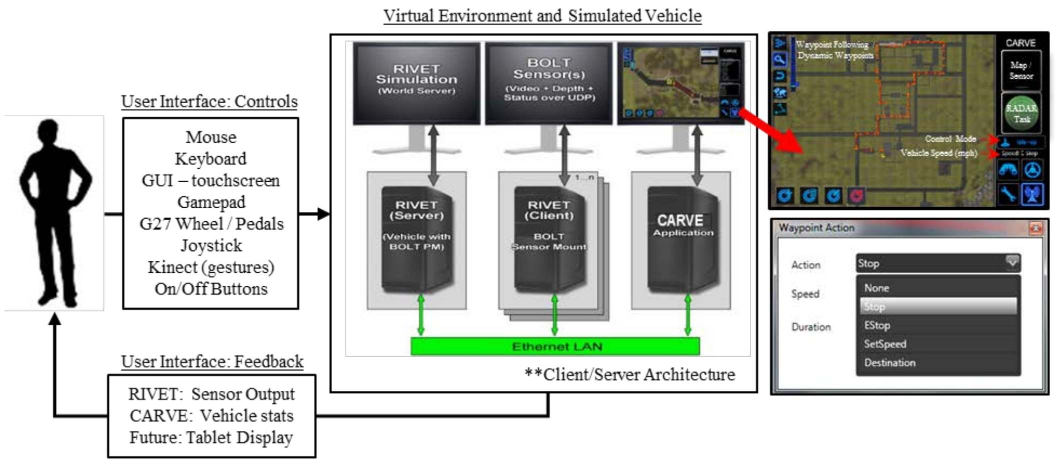


Fig. 3 Customizable simulation environment for HRI (RIVET, BOLT, CARVE)

Additional descriptions and capabilities of this software, as well as specific modifications for HRI experimentation, will be discussed through Section 4 (Procedures for Using RIVET for HRI Experimentation). We note here that CARVE may not be currently set up to meet every possible need for HRI experimentation; however, additional user interfaces can be programmed to function within CARVE. This may require more advanced knowledge of computer programming, specifically in C#.

4. Procedures for Using RIVET for HRI Experimentation

The design and development of RIVET, and the associated CARVE application, are valuable tools for HRI experimentation. In this section, we provide key descriptions of the components and customization capabilities of this simulation system. As such, RIVET and CARVE can have a wide range of HRI operations, from combat-specific operations (e.g., surveillance, cordon, and search) to base operations (e.g., passenger transit using a self-driving vehicle). However, different types of customization require varying levels of computer programming knowledge, which are noted throughout the following sections. Step-by-step directions for using RIVET are then provided in the appendixes.

4.1 Initial Research Definition

Before starting the mission setup, the HRI experimenter should have a research question, a set of robot system requirements, and an application. Based on this information, it is beneficial to create a storyboard of the scenario or use case (Preece et al. 2015). This will help the experimenter outline the necessary elements that will be needed in the development of the VE. This may include the location (e.g., Middle

Eastern town versus US military post), terrain and weather specifications, potential risks, stationary and dynamic objects, anticipated or actual robot capabilities and behaviors, as well as the team goals and objectives. Directions on how to create a new mission environment are provided in Appendix C.

4.2 Setting up the Mission

4.2.1 Virtual Terrain

The first step in developing the VE in RIVET is to build the virtual terrain. The current setup and components of RIVET are such that a user with a basic computer science background has sufficient knowledge to develop the virtual terrain. When starting to develop a new virtual terrain, we first recommend exploring the missions that already exist in RIVET. Available mission files can be accessed through the Main Menu (Fig. 4) by choosing a file from the Loading Mission screen (Fig. 5). This will allow HRI experimenters the opportunity to see what environments have already been built and could be leveraged for their needs.

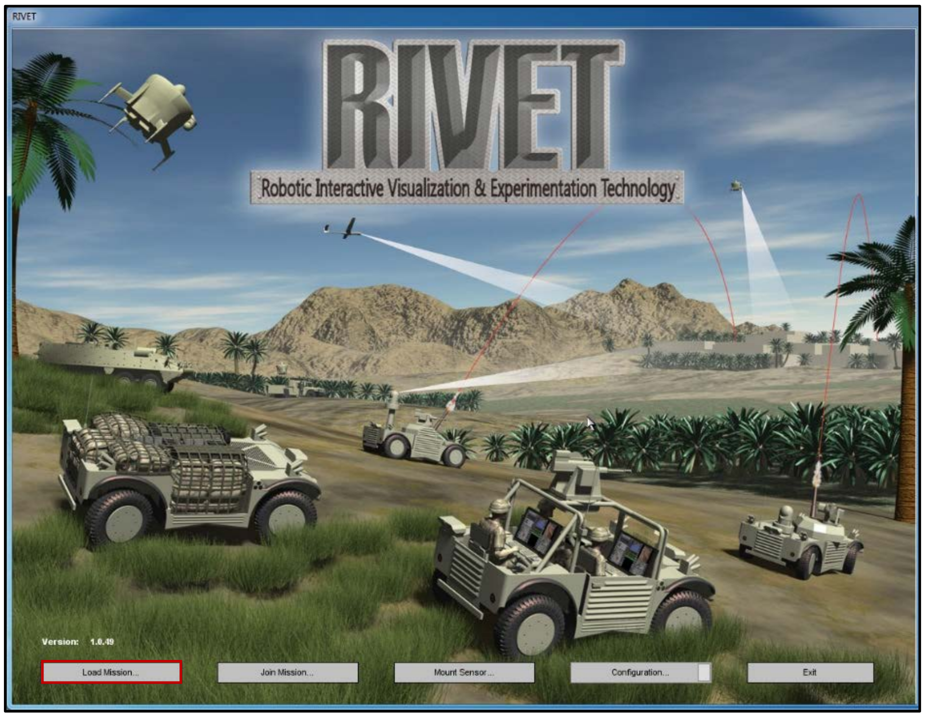
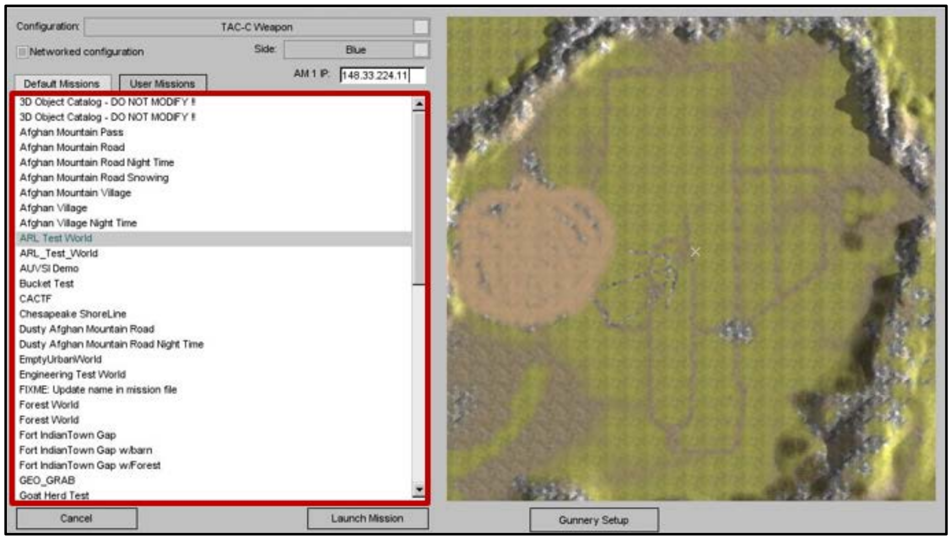


Fig. 4 Main Menu screen



Configuration: TAC-C Weapon

Networked configuration

Side: Blue

AM 1 IP: 148.33.224.11

3D Object Catalog - DO NOT MODIFY !!
3D Object Catalog - DO NOT MODIFY !!
Afghan Mountain Pass
Afghan Mountain Road
Afghan Mountain Road Night Time
Afghan Mountain Road Snowing
Afghan Mountain Village
Afghan Village
Afghan Village Night Time
ARL Test World
ARL_Test_World
AUVSI Demo
Bucket Test
CACTF
Chesapeake ShoreLine
Dusty Afghan Mountain Road
Dusty Afghan Mountain Road Night Time
EmptyUrbanWorld
Engineering Test World
FIXME: Update name in mission file
Forest World
Forest World
Fort IndianTown Gap
Fort IndianTown Gap w/barn
Fort IndianTown Gap w/Forest
GEO_GRAB
Goat Herd Test

Note: The red box highlights the areas related to current default missions that have been previously created.

Fig. 5 Load Mission screen

It is possible that none of the available mission environments are a direct match for the current research goal. If this is the case, it is still possible to begin with a base environment and customize it to meet the specific needs of the mission. The ARL Test World is a recommended option to start to develop a new mission file. It has a relatively blank canvas, containing only a Soldier following a path and a small building. After entering the VE, it is possible to adapt the terrain to match the needs of the mission. The terrain is sculpted using the RIVET terrain editor tools. These tools permit changes in the appearance of the terrain using different textures that represent rocks, gravel, grass, dirt, or the forest floor. Instructions for adapting the terrain are provided in Appendix D. More artistic models, such as grass and trees, are also available by adding static objects.

4.2.2 Static Objects

The TGE allows the programmer to create realistic simulated terrains that can be textured and shaped to give an appearance of any setting desired. To illustrate the complexity that can be created with the editing tool, Fig. 6 shows a virtual Middle Eastern market place with streets, stalls, and buildings. Each one of these items must be carefully positioned to reflect a marketplace in a developing country. Depending on complexity, a scene may contain hundreds or thousands of objects. Making a scene look professional may require a lot of work and time, but the realism may aid testing and the capability to transition findings to the field.

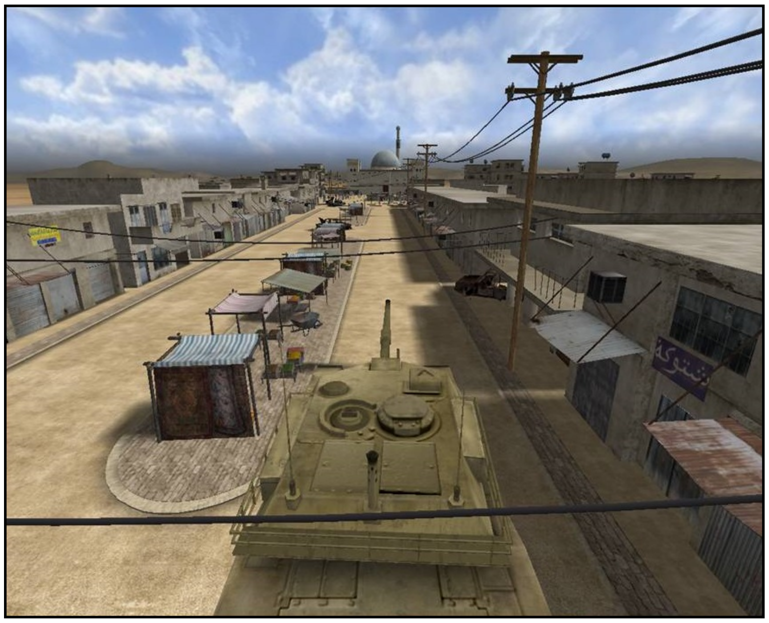


Fig. 6 Complex world building

The TGE has built-in tools that allow the programmer the ability to tailor the scene. The World Editor is accessed by pressing the F11 key on the keyboard. Figure 7 shows the marketplace objects augmented with entity names shown in white. In this view, each yellow dot represents a different object, such as a building, burned-out car, and even the electrical lines. This is where objects can be added, deleted, or moved along each axis (see also Appendix E). Each object may also be rotated in any direction along each axis. Notice that many of the objects are tagged with the "null" label. This means that the object has not been assigned a specific name during the creation of the scene. Naming each object is possible through the TorqueScript files, the console, or the World Editor Inspector. The purpose of the World Editor Inspector is to allow the evaluator to inspect and modify parameters for individual objects. This is where the position, rotation, scale, name, and many different dynamic fields can be changed. These tools are essential to building a realistic environment.

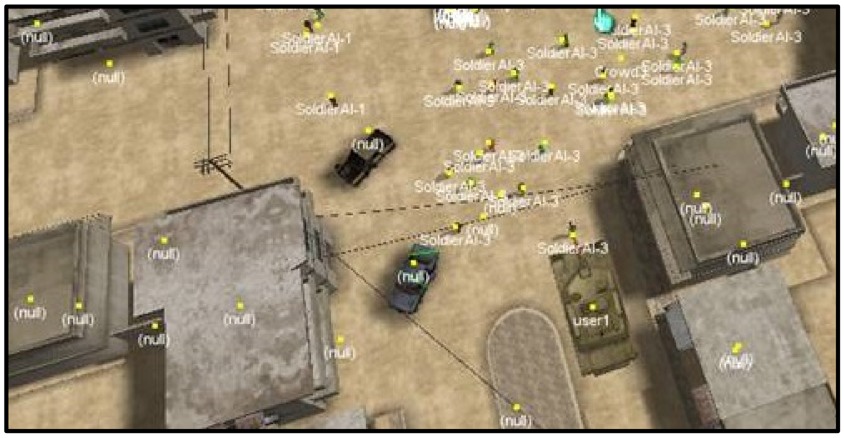
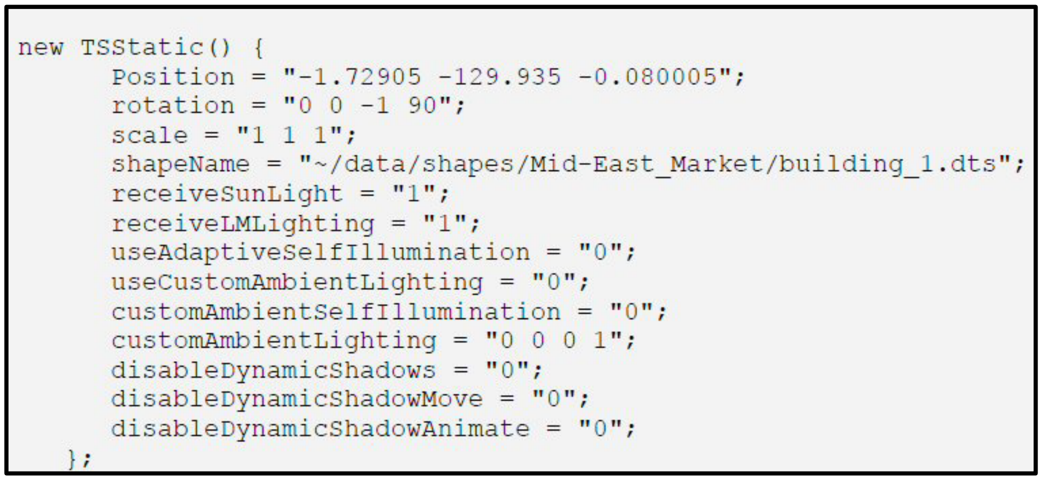


Fig. 7 Example of the view from the World Editor Inspector

All objects within the scene are defined in a mission file, which is saved as a text file. Each object is listed along with its respective parameters. A static object is defined in the mission file as shown in Fig. 8.



```
new TSStatic() {
      Position = "-1.72905 -129.935 -0.080005";
      rotation = "0 0 -1 90";
      scale = "1 1 1";
      shapeName = "~/data/shapes/Mid-East_Market/building_1.dts";
      receiveSunLight = "1";
      receiveLMLighting = "1";
      useAdaptiveSelfIllumination = "0";
      useCustomAmbientLighting = "0";
      customAmbientSelfIllumination = "0";
      customAmbientLighting = "0 0 0 1";
      disableDynamicShadows = "0";
      disableDynamicShadowMove = "0";
      disableDynamicShadowAnimate = "0";
   };
```

Fig. 8 Example TorqueScript for a static object located within the Mission File

4.2.3 Dynamic Objects: Vehicles

There are many assets developed for use within the simulation (Fig. 9). The vehicles can move about using direct user control through keyboard or joystick, script-based path following, or file-based waypoint following using either time on target or locations and velocity data. The unmanned vehicles consist of entities from the following classes:

- Unmanned ground vehicles
- Unmanned air vehicles
- Unmanned surface vehicles

- Unmanned underwater vehicles
- Small unmanned ground vehicles

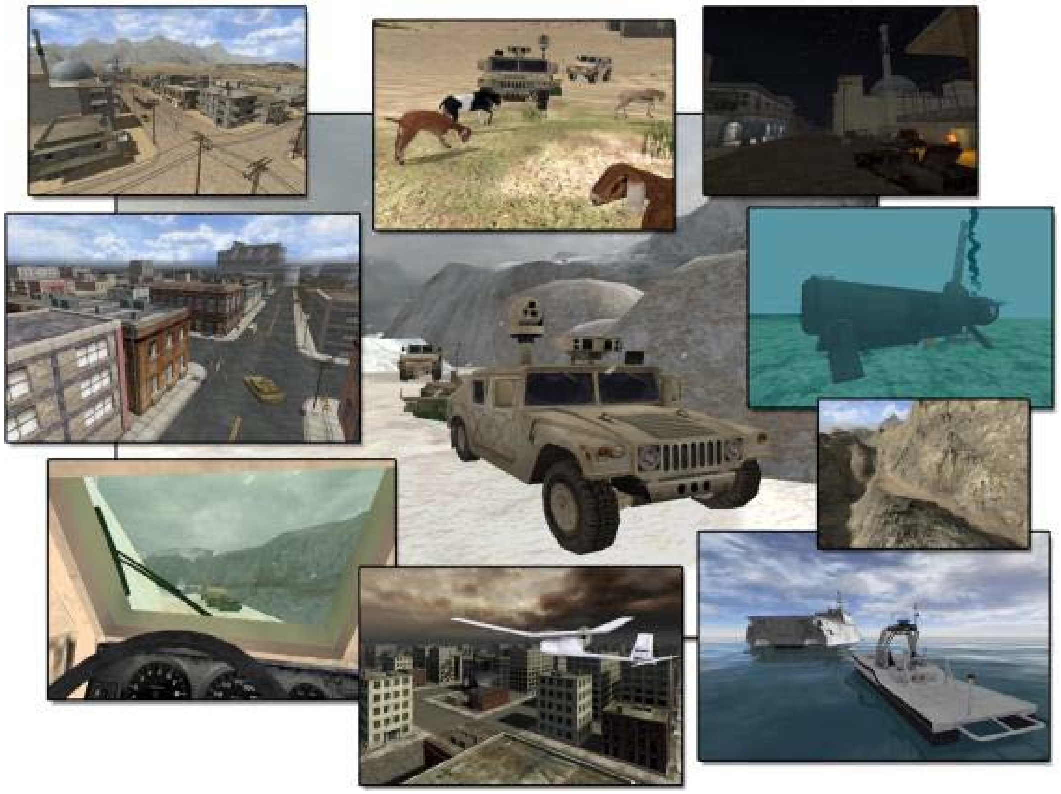


Fig. 9 Examples of unmanned systems available within RIVET

Specific missions may also require additional non-player vehicles to have preprogrammed movements throughout the mission space. Here, the term non-player is included to differentiate between vehicles that will be autonomously moving in the VE from player vehicles that a participant has an option to control. The current version of RIVET has some path-following vehicles already available (Fig. 10). It is important to note that these dynamic elements used in a scenario require code to control their actions. Animations are created within modeling tools such as Autodesk 3ds Max, and controlled with script to allow movement of objects. TGE uses a scripting language, aptly named TorqueScript, to provide the control of these dynamic elements. It is beyond the scope of this report to provide an in-depth discussion of TorqueScript. The works of Maurina (2006) and Finney (2007) provide more information.

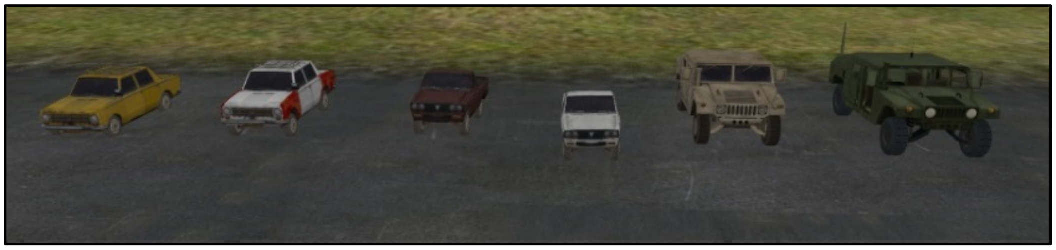


Fig. 10 Examples of path-following vehicles in RIVET

Step-by-step directions on adding a path and a path-following vehicle to the mission file are available in Appendix F and Appendix G, respectively. With these directions, the vehicle will appear by default when the mission file is opened. The vehicle will continue on the path provided. If the mission file is saved, the vehicle will begin at the last saved point the next time the mission file is opened.

4.2.4 Dynamic Objects: People

Human characters are commanded using direct user control, scripted path following, or crowd logic. Similar to the above descriptions on path-following vehicles, it is also possible to have path-following people, also referred to as non-player characters, or NPCs. These NPCs currently include civilians, Soldiers, and enemy combatant characters. It is also possible to *trigger* the NPC to spawn (also known as the creation or appearance of a character) on the path at a set point in time, as well as trigger the NPC to disappear at a set time and location. A trigger is "a volume of space that initiates script callbacks when an object enters, stays inside, or leaves the trigger's volume" (http://docs.garagegames.com/torque-3d/reference/classTrigger.html#_details). More specifically, a trigger causes an event to happen. Having the option to create or delete the NPC allows the experimenter more flexibility and consistency between runs or trials. However, this requires additional computer programming, as well as additional TorqueScript files. Step-by-step examples, including the triggers to create and delete path-following Soldiers are provided in Appendix H and Appendix I, respectively.

At times, it is more beneficial to use the crowd logic than to create multiple paths for many individuals. Figure 11 shows a flurry of activity among Soldiers and civilians. These crowds were generated using the Crowd Editor, a tool designed to add groups of simulated NPCs to specified regions of the VE. Within this region, the people move, interact, and mill about simulating a realistic marketplace. Directions for adding crowds are provided in Appendix J.

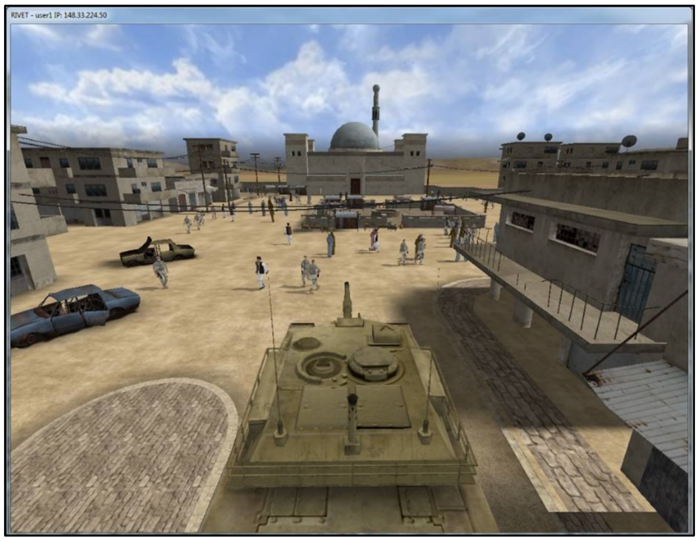
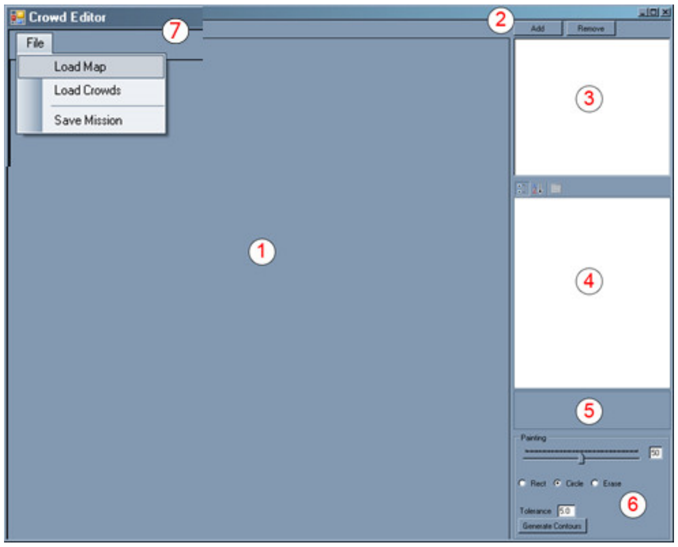


Fig. 11 Example of crowd modeling in RIVET

The RIVET Crowd Editor is a C# application that allows a user to add groups of NPCs to a mission through a GUI. The GUI allows for configuring, adjusting, and controlling these simulated crowds. The editor is used to add crowd entities with a specific number of objects, their type, and the percentage of each character type. The editor also has the feature where the user chooses the number of destinations and how long the NPCs are idle once they reach their destination. The Crowd Editor GUI is shown in Fig. 12.



[1] **Map View** shows all the visual crowd data that is being applied to the selected map.
[2] **Add** and **Remove** is used to add and name a crowd as well as to delete it.
[3] **Crowd List** provides a list of all the crowds created for the selected map.
[4] **Property View** enables you to configure and edit all the aspects for the specific crowd selected from the **Crowd List**.
[5] **Description Window** shows a description of the selected property in the **Property View**.
[6] **Tool Box** supplies all the tools used to create and generate the crowd.
[7] **Menu** allows user to load a map, load crowds, or save a mission.

Fig. 12 Crowd Editor GUI in RIVET

4.3 Robot Autonomy

Up until this point, options and modifications that are specific to RIVET have been described. However, scenarios using advanced vehicle or robot autonomy require the integration of the BOLT sensor and the CARVE application. Since missions can require a user to intervene and “drive” a vehicle, it was important to devise a way to communicate with the vehicle in the simulation. The BOLT interface was developed to allow communication over the network. The User Datagram Protocol/Internet Protocol (UDP/IP) is used for communication outside of the application. The BoltPMmount code in the game engine allows access through UDP/IP. The BOLT code allows the CARVE GUI to communicate through to the game engine. Commands are sent from the GUI to the vehicle, directing movements based on inputs from the code, keyboard, joystick, or steering wheel.

The platform mobility code provides access to the vehicle information while the BOLT sensor provides access to the camera buffers. The configuration menu on the RIVET startup screen allows for construction of vehicles, as well as sensors. To use a vehicle in the application, a vehicle is first selected, named, and then sensors are added. A separate computer running RIVET can mount a BOLT sensor to connect to the server computer. With a vehicle properly configured with BOLT, the CARVE application is able to connect to RIVET. Directions on how to connect to CARVE are provided in Appendix K.

4.3.1 Teleoperation, Remote Control, Manual Control

CARVE is designed to work with a number of controllers, which provides increased flexibility in the experimental design. Adapting the CARVE program code to set the controllers may require a more in-depth understanding of computer programming. However, it is possible to customize a variety of controllers, including the following: keyboard, mouse, touch screen GUI, Logitech Gamepad, Logitech joystick, Logitech G27 wheel and pedals, Microsoft Kinect, and other user interfaces (e.g., engage/disengage buttons).

The original controllers available for use with CARVE were the keyboard and mouse, a Logitech gamepad, and a touch screen track ball. The keyboard and mouse controls were designed to mirror the user interface and key commands described in the *RIVET Developer’s Guide* (General Dynamics Robotic Systems 2010, 48). The Logitech gamepad controls were designed to provide additional fidelity for interaction with a vehicle. In addition, each button is programmable, and can be used for dual-task experimentation. Figure 13 provides a visual representation for the current Logitech setup.

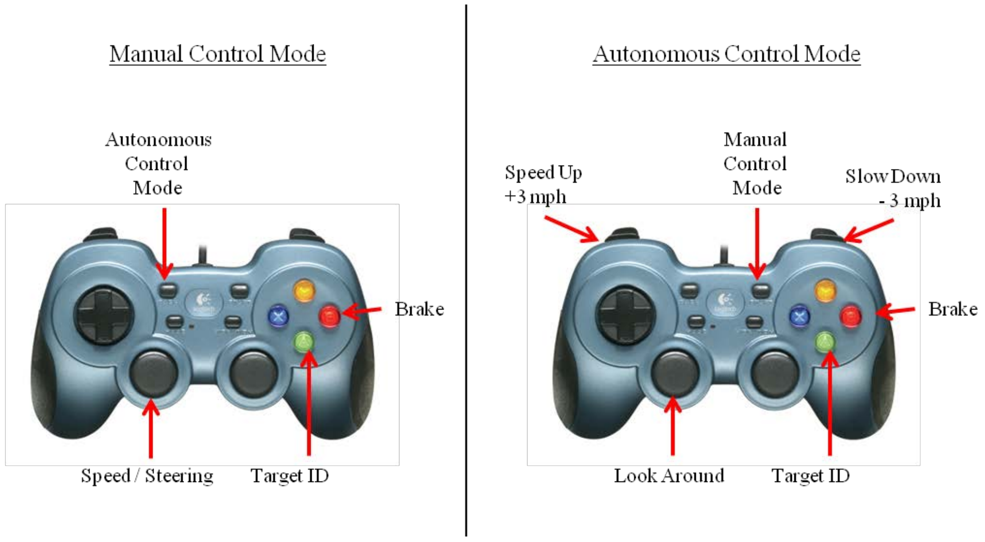
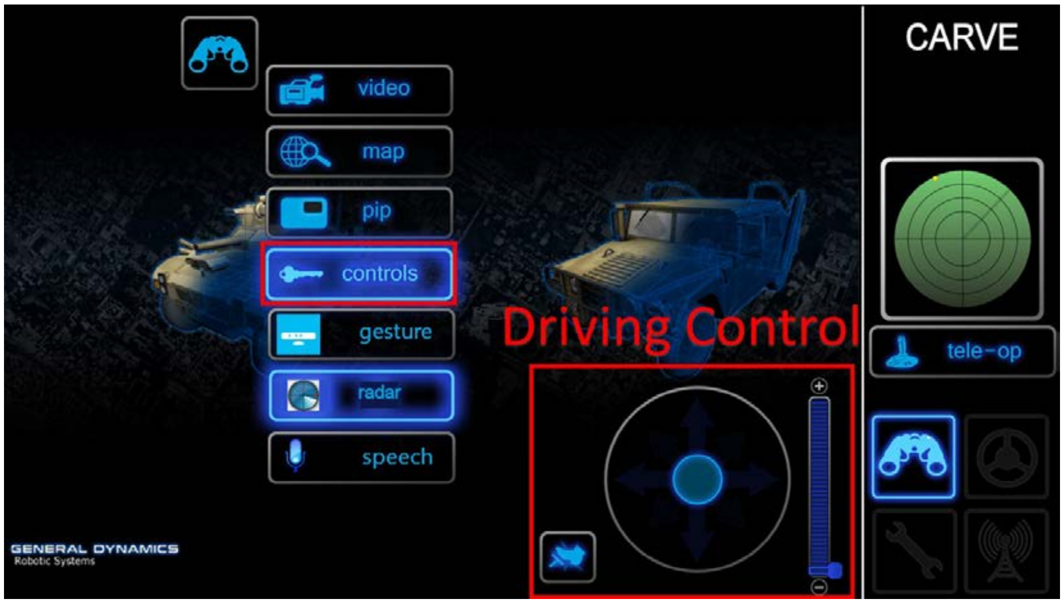


Fig. 13 Logitech gamepad controller current button layout

The touch screen track ball controls for speed and movement are directly on the CARVE application screen (Fig. 14). This controller interface is very similar to a joystick. The user will move the circular thumb pad to drive the vehicle in the direction the center ball is moved. The farther from center the ball moves, the faster the speed at which the vehicle moves.



Note: The red text and outline are of buttons are used here to demonstrate key parts of the GUI for demonstration purposes only. They are not part of the actual GUI.

Fig. 14 Touch screen track ball for driving controls

To meet the diverse needs for a variety of experiments, CARVE has been updated with the capabilities to use the Microsoft Kinect sensor, the Logitech G27 Racing Wheel and Pedals, as well as additional engineered controllers (e.g., button activation controls). These additional user inputs allow for more experimental control.

The Kinect sensor is a low-cost (~$150) video and depth sensor with an open source software development kit that allows developers to use the device for gesture input, as well as spoken language (Fig. 15).

Fig. 15 Microsoft Kinect sensor

To update the CARVE for manual and autonomous self-driving vehicle capabilities, controls for the Logitech G27 Racing Wheel were also added to the CARVE software. The current functionality allows for full manual control driving. In addition, an automation button system was built and programmed to engage and disengage the vehicle's automation. These controllers, when used together, allow a user to switch between autonomous self-driving vehicle control and manual control (Fig. 16).

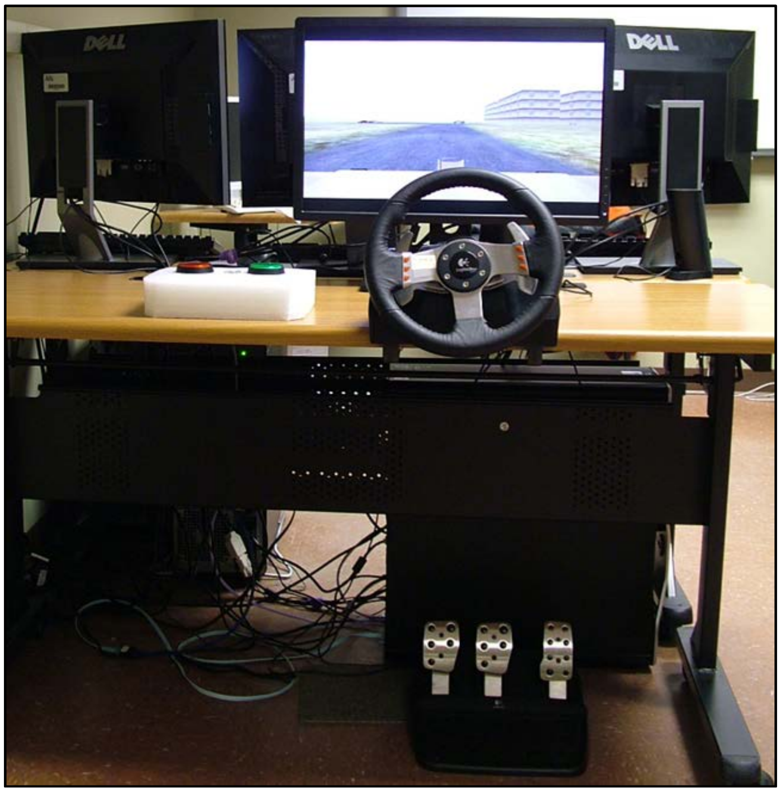


Fig. 16 Logitech G27 racing wheel and pedals, and additional button controls

4.3.2 Waypoint Following

CARVE was designed to integrate waypoint following to give the vehicle a level of semi-autonomy for navigation (Fig. 17). Here, waypoints are markers, represented by an orange circle, that are placed in set locations for path planning (navigation) prior to the start of a mission. Using waypoint following allows the user to conduct additional tasks such as watching a video stream or watching a radar screen for potential enemies. Waypoint paths are created on the map screen (Fig. 17) using the Add Waypoint button [A]. Each click adds an additional waypoint to the path. The entire path can be deleted using the Delete button [B]. Once a path is complete the Change button [C] is used to adjust waypoints on the screen.

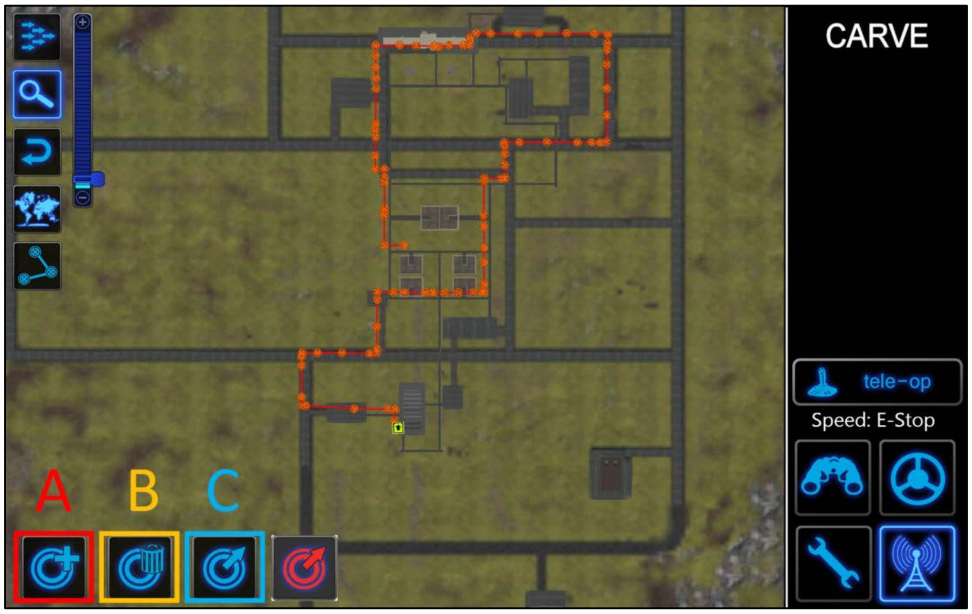


Fig. 17 Waypoint following in CARVE

4.3.3 Autonomous Capabilities

While the original waypoint-following capabilities mirrored real-world semi-autonomous navigation through the placement of different waypoints, it was important to advance the capabilities of the system to include a customizable control system. For this purpose, dynamic capabilities were added to the waypoints (Fig. 18). Step-by-step directions for setting dynamic waypoints are included in Appendix L.

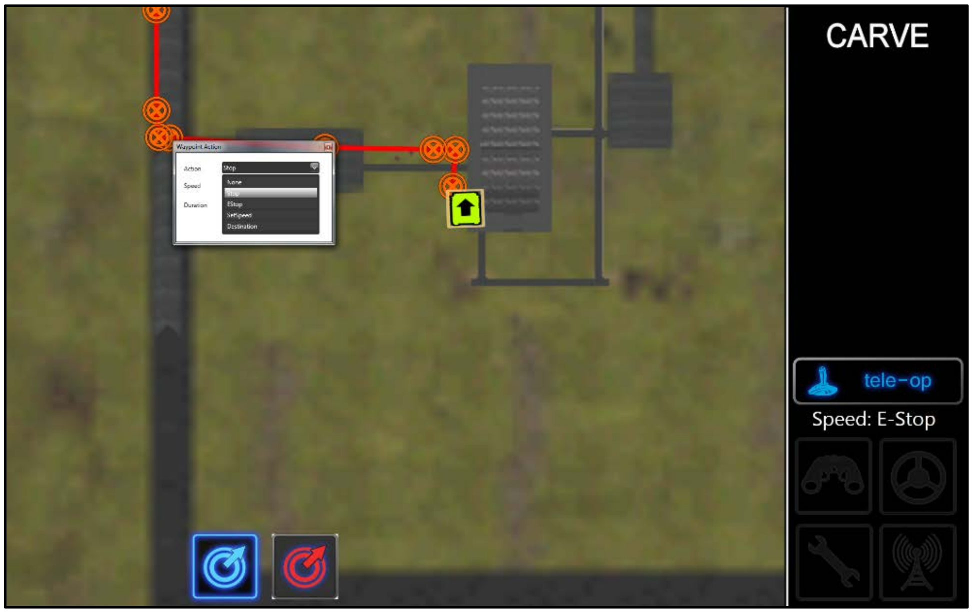
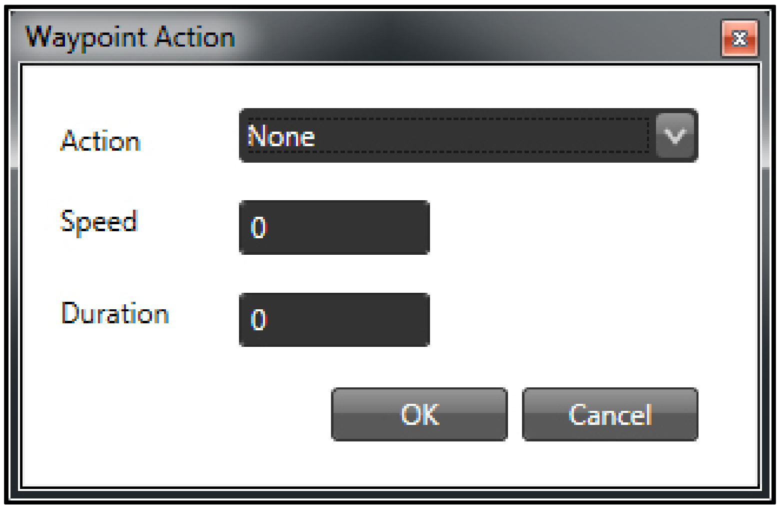


Fig. 18 Options for dynamic waypoints in CARVE

A feature that was added to the latest release of CARVE was to allow adjustment of actions between waypoints. Upon adding a waypoint to the map, the Waypoint Action dialog box opens and an action can be selected from the list (Fig. 19 and Fig. 20). If “None” is selected, then the same specifications from the previous waypoint are kept. This option can be used for “navigation only” markers. If “Stop” is selected, then the vehicle will stop at that waypoint for the “duration” set in the last text box. Duration time is set in milliseconds. If “Estop” is selected, the vehicle will remain stopped until the user releases the vehicle by re-engaging the automation through a button press. This capability allows the experimenter to set breaks in the simulation. The “SetSpeed” selection is coupled with the “Speed” entered. The speed is set in miles per hour (mph). Finally, the “Destination” option functions similar to the “None” in that it provides a navigational marker. However, it is has a different link to an output file that differentiates between these points. Therefore, it can be used to record time stamps for specific events.


Waypoint Action

Action: None

Speed: 0

Duration: 0

Fig. 19 Waypoint Action menu

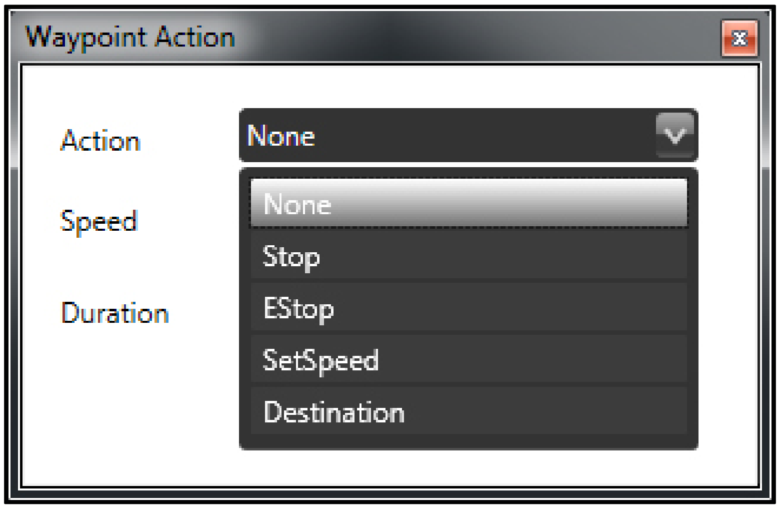


Fig. 20 Dynamic waypoints available actions

4.4 Alternative Tasks

Additional tasks that participants can complete can also be added to the CARVE interface. These customizable tasks can be used for a dual-task paradigm or to increase the participant's workload. One such example is the radar task (Fig. 21). For this task, participants will monitor the radar sensor and identify when a blip (small colored dot) changes from yellow (target) to red (threat) by pressing a button on the user control interface. Once the threat is identified, the blip will turn from red to blue providing feedback to the participant that their task was recorded. CARVE is set up to record performance on this task to a .csv file. Performance metrics include correct detection, false alarms, misses, and detection time. This task is also customizable to create conditions of varying workload (e.g., number of possible targets, time between targets, speed of movement, or pattern of movement). Instructions are provided in Appendix M.

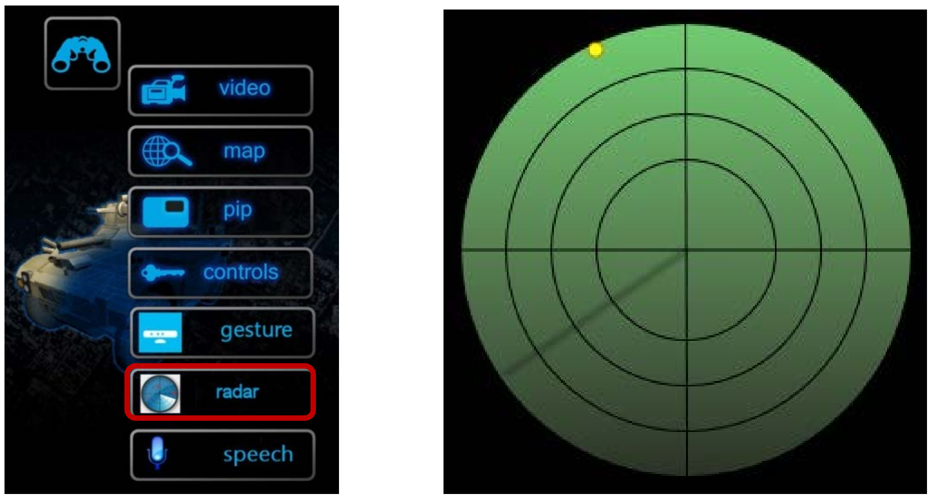


Fig. 21 Alternative radar target detection task in CARVE

4.5 Manipulating and Measuring HRI

RIVET and CARVE provide a number of customizable capabilities for manipulating and measuring HRI. This section provides a brief review of some of those capabilities for data collection as well as references for others. These may include recording the behaviors of the simulated robot or virtual Soldier avatar to determine frequency and duration of HRI, or data for use in an after-action review (AAR). An AAR is a review of the simulation experiment after the fact, where participants can provide opinions and feedback on their personal assessment of the experiment. Such AAR data can supplement quantitative data collected from the simulation and interaction. Depending on what data from the simulation or the interface are needed, the code base can be developed to gather and export that information to an output file through RIVET or CARVE.

RIVET has a capability of recording data from the game engine and the scene. This is done using code in the engine to allow for Structured Query Language (SQL) calls to capture, record, and manage data held in a relational database. For more information on how to read, write, and append custom files, see FileIO Tutorial (http://docs.garagegames.com/tgb/official/) or the section of the Game Programmer's Guide to Torque on writing and appending files Section 9.5.9 Writing and Appending Files of the Game Developer's Guide to Programming Torque (Maurina 2006, 370).

There are also ways of recording data using the CARVE interface. A class was created named "WWDataLogger", which was used to gather data from user interaction with the experiment. In this example, all of the data is gathered and written to a Microsoft Excel file. It records the location of the target blip, the activity (path started, enemy in zone, enemy identified, enemy misidentified, and path completed), and a time stamp. From this, it is possible to calculate signal detections, which are delineated as correct response, errors (misses, misidentifications), and detection time. Data collection can also be customized to record robot behaviors and number of interventions in the robot's autonomy with recording of an event, time, and associated data (e.g., duration and associated times of button presses, type of button, as well as time and frequency between manual and autonomous modes). This type of data collection requires more advanced programming to create the output file (.csv or Microsoft Excel). Depending on what data from the simulation or the interface are needed, the code base can be developed to gather and export that information into a separate file.

Customizations to the simulation environment can be made in line with the HRI metrics by Steinfeld et al. (2006), Goodrich and Schultz (2007), and Weiss et al. (2009), as previously described in Section 2.1 of this report.

- *Navigation* of the simulated robot: The effectiveness (e.g., number of obstacles avoided or tasks completed) and efficiency (e.g., time and duration) related to navigation are elements that can be recorded in RIVET (.sql), CARVE (.csv), or by hand during an experiment (e.g., pen and paper). Participant feedback through an AAR or video analysis can be accomplished by recording the mission run for later viewing through the integration of additional video recording software, such as Fraps (www.fraps.com).

- *Perception* from sensor data: RIVET provides simulated sensor readings that come in the same format that a user would receive on the real robot. Sensor data can then be run through perception algorithms to identify how well the robot understands the world around it. Current configurations of the simulated robot can include camera, lidar, and semantic sensors (Gonzalez et al. 2009). The details for collecting and assessing sensor perception data are outside the scope of this report. However, for more information, Dean and DiBerardino (2014) provide resource material on the integration of the Common World Model.

- *Management* and function allocation: Since this metric area is related to user interfaces and control, we recommend using customizable data recording through the CARVE application. It is possible to set the frequency, duration, type, degree of movement, etc. for every intervention using a user control interface to record into a .csv output file. However, this may require more advance programming capabilities.

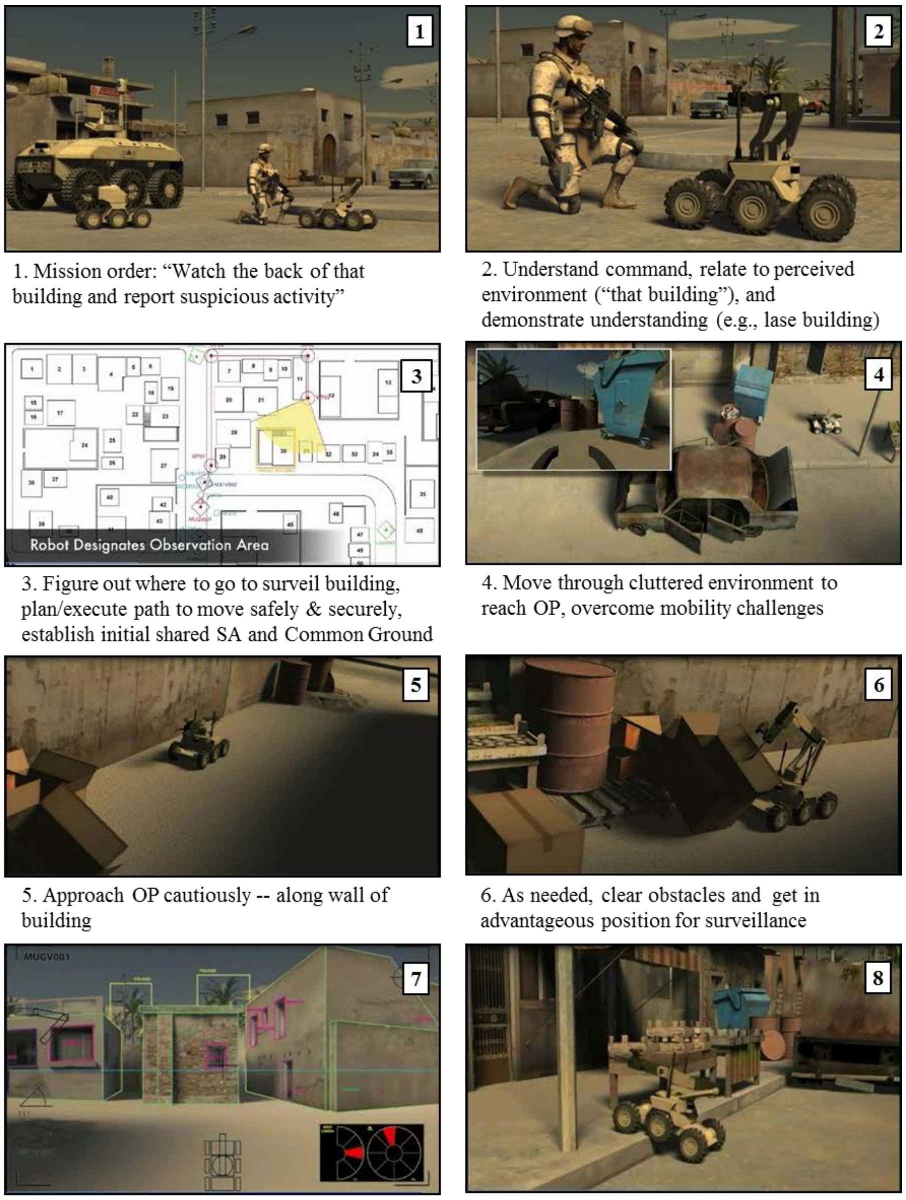
- *Manipulation*: Here, manipulation is referring to how the simulated robot is directly influencing the VE. RIVET offers some degree of manipulation required for HRI type tasks; however, fine-grained manipulation required by a real-world robot is nearly impossible to accurately model in this simulation environment. This is because RIVET is built on a game engine instead of in a high-performance computing simulation. However, assessing HRI may not require fine-grained manipulation. Therefore, we recommend that experimenters refer to the design guidelines and the requirements documents of the real-world robot. Programmable animation sequences can then be built and integrated into the VE (General Dynamics Robotic Systems 2010, 67–232).
- *Social interaction*: There are a number of opportunities for manipulating and recording social interaction within RIVET and CARVE, as well as with obtaining feedback after the fact with an AAR. These measures include, but are not limited to, recorded distances between people and robots, manipulation of behaviors, communication exchanged (e.g., programmable feedback system), programmable team roles, as well as usability, social acceptance, and individual differences. These metrics can be recorded through the development of customizable GUIs, programmable pauses, recordable behaviors, and even integration of outside tools (e.g., eye tracking).

5. Previous HRI Research Using RIVET

A number of research efforts through the RCTA have used various functionalities of RIVET and CARVE for HRI experimentation. One of the main goals of this type of experimentation was the advancement of robot design through the consideration of the human or the human-robot team. As mentioned earlier in this report, the research discussed here is in line with ARL's Intelligent Systems Enterprise vision that enables the teaming of autonomous, intelligent systems with Soldiers. This section discusses experiments in both dynamic and unstructured combat environments, as well as noncombat military installations and base operations. For additional resources on similarities and differences between RIVET and field experiments in general, see Bodt et al. (2010) and Schafer et al. (2015). This section demonstrates the diversity and customizability of RIVET, which make it a valuable platform for HRI experimentation.

5.1 Cordon and Search

The cordon and search task has been used as a motivating scenario by ARL researchers. During urban transit by a small (4–5 Soldier) unit, a fugitive is reported to have entered a building that the unit is approaching. A human-transportable robot is instructed to “screen the back door” of the building by the unit commander since he cannot safely split up his limited resources (Fig. 22). While this narrative occurs in the context of a cordon and search operation, its underlying capabilities support a broad range of potential operational missions.



1. Mission order: “Watch the back of that building and report suspicious activity”

2. Understand command, relate to perceived environment (“that building”), and demonstrate understanding (e.g., lase building)

3. Figure out where to go to surveil building, plan/execute path to move safely & securely, establish initial shared SA and Common Ground

4. Move through cluttered environment to reach OP, overcome mobility challenges

5. Approach OP cautiously -- along wall of building

6. As needed, clear obstacles and get in advantageous position for surveillance

7. Assess egress points, detect humans, update status

8. Upon mission completion, rejoin unit, maintain SA, accept new missions

Note: OP = objective point.

Fig. 22 RCTA motivating scenario (ARL Robotics Collaborative Technology Alliance 2012)

5.1.1 Target Detection: Person

In this experiment, participants monitored a Talon robot complete the cordon and search type task to monitor the back of a building for human terrorist targets (Schaefer 2013, 2016). A single computer (RIVET only) setup was ideal since participants viewed the Talon robot's camera sensor from a remote location and were unable to directly intervene in the robot's behavior. For the purpose of this experiment, a single video of the simulation was created from the camera-view on the Talon robot that showed the robot navigating to the back of a building, finding a secure location, and monitoring the back door for human targets. The video of the simulation was created using Fraps real-time video capture and benchmarking program with a 30 frame/second (fps) .avi file. The .avi file was converted into the .mp4 file format to add auditory feedback from the Talon robot in post-processing. This allowed the experimenter to create multiple conditions from a single simulation. Condition 1 provided a 100% reliable detection condition in which the Talon robot provided verbal feedback (i.e., "target detected") for all the human targets. Condition 2 provided a 25% reliable detection condition. While auditory feedback could have been added within RIVET, it would have required the creation of separate mission files. Thus, use of outside software was ideal.

During development, the VE used was a previously developed base environment named CACTF (Combined Arms Collective Training Facility). This VE was originally developed to represent a simulated version of a real-world training facility of an urban environment that was designed to conduct multi-echelon, full-spectrum operations training up to battalion task force level. The base VE included the layout of the physical environment (e.g., ground, roadways, buildings, and lighting). To meet the needs of this mission, terrorist NPCs were created and added to RIVET (Fig. 23). More information on creating NPCs, animations, and adding to the TGE can be found in the *RIVET Developer's Guide* (General Dynamics Robotic Systems 2010).

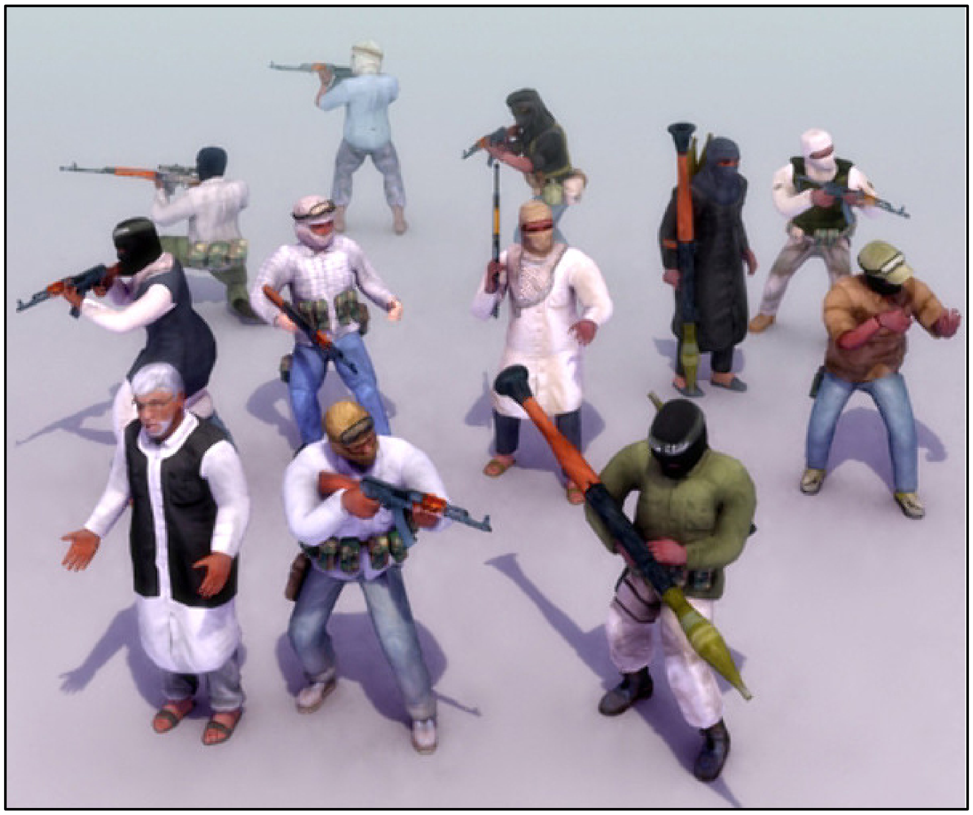


Fig. 23 Terrorist NPCs

Additional capabilities were added to RIVET to meet the needs of this experiment. Task-specific customization of the environment was accomplished through creation of RIVET GUIs, as well as Scripting syntax using Torsion. Torsion is a powerful integrated development environment for creating TorqueScript-based games and modifications. Specific customization included entering objects, obstacles, and creation of paths, to name a few. For customizable control of the Talon, additional script files were included to create and set up a path-following Talon robot (see Appendix F and Appendix G for direction on adding a path-following vehicle). This allowed a preprogrammed path and the robot behaviors to be set before recording the mission. This addition to RIVET has resulted in a number of vehicles (e.g., Humvee, taxi, KBot, Talon) with path-following capabilities that are available for future simulation experiments.

5.1.2 Target Detection: Object

This experiment used a single-user format (one RIVET computer) and was designed without adding any new technology or requiring any additional TorqueScript files. For this experiment, participants monitored a computer agent (either a Soldier avatar or a Talon robot) as they located 16 missing M16 rifles throughout a Middle Eastern town (Sanders et al. 2012). Two videos of the simulation (one for a 100% reliable agent and one for a 25% reliable agent) were created from the third person perspective for each agent navigating through the environment and locating targets.

The VE used a base environment of the Middle Eastern Marketplace that already had static and dynamic objects, and a crowd of people. To meet the needs of this mission, a path-following Soldier and a path-following Talon robot, as well as the missing rifles were added to the environment (Fig. 24). This allowed additional control in navigation path and timing between the agents. The video of the simulation was created using Fraps real-time video capture and benchmarking program with a 30 fps .avi file. For the purposes of this work, an external physiological measurement tool (Labscribe; iWorx, Inc.) and an eye tracker (EyeWorks; Eyetracking, Inc.) were used to collect physiological and behavioral data. Results are provided in Sanders et al. (2012).

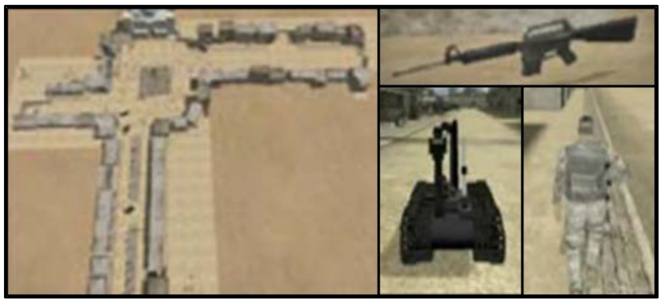


Fig. 24 Example of RIVET map, targets, and avatars (Talon and Soldier)

5.1.3 Navigation

For this experiment, participants controlled a Soldier agent situated within a Middle Eastern environment (Mid East RCTA Demo). The task was to assist an autonomous Talon robot from a set location to a rendezvous point as quickly as possible (Schaefer 2013, 2015). This assistance required the participant to locate the robot and move obstacles out of the path if the robot was unable to navigate around these obstacles. Participants completed this task with 2 different robots. The first robot was 80% reliable in self-navigating obstacles (successfully navigated around 4 out of the 5 potential obstacles). The second robot was only 20% reliable (successfully navigated around one obstacle).

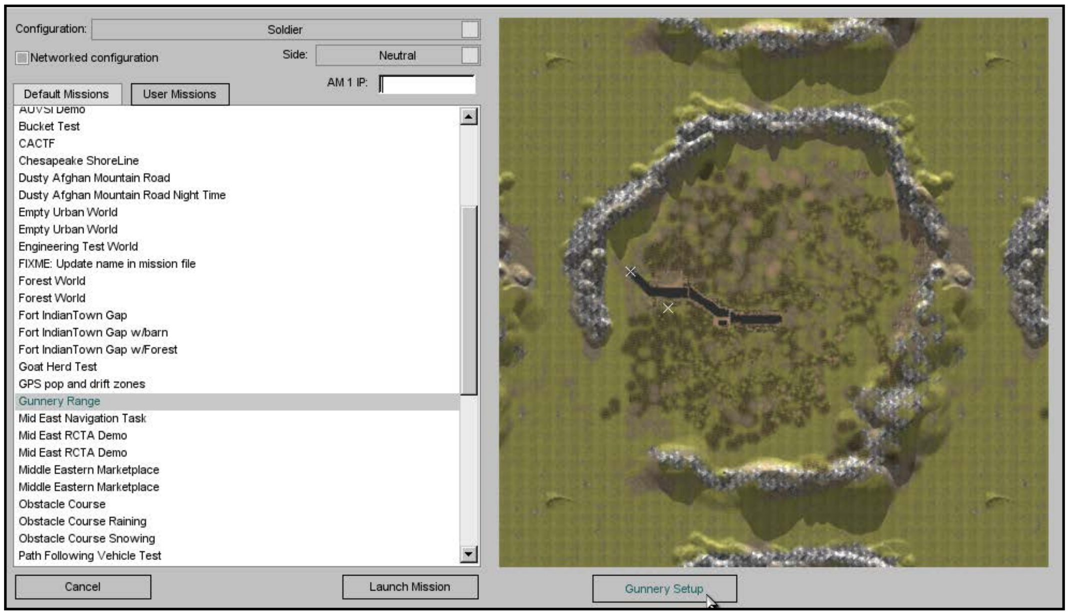
Due to the design of this experiment, moveable objects were needed and created by General Dynamics Robotic Systems (GDRS) for use in RIVET. This included syntax for a refrigerator, box, crate, barrel, and trashcan (refer to C:\RIVET\sim.base\server\moveableobjects.cs). In addition, independent task-specific customization of the physical environment was accomplished through TorqueScript syntax. Specific customization included entering objects, obstacles, creation of paths, etc. Scripting files were created for both the training session and the task conditions. The simulated tasks were recorded using Fraps real-time video capture and benchmarking program with a 30-fps .avi file. Video was recorded from the Soldier character's perspective for later data analysis.

5.1.4 Transparency

Transparent bi-direction communication is an extremely important component to effective and trusted human-robot teaming (Barnes et al. 2016; Schaefer et al. 2016). To address this HRI issue, Sanders et al. (2014) used RIVET as a single computer architecture for a joint Soldier-robot team responsible for clearing an area of weapons and locating civilians by marking their location on a map. The authors did not require any additional functionality from RIVET for this task. They did, however, integrate the RIVET simulation with E-Prime to adapt the type of communication from the robot. This included variations in the mode of communication from the robot (e.g., audio, text, graphics) and the amount of information being communicated (e.g., minimal, contextual, constant).

5.2 Gunnery

A second type of combat-related mission was created in RIVET to conduct virtual unmanned gunnery exercises. The purpose was to assess the effect of various interface devices for robotic asset control on system operation and overall performance. A series of gunnery exercises was created to assess the ability of the operator to identify targets, select the appropriate weapon, and engage targets with the handheld gamepad and Microsoft Kinect voice commands. This experimental setup required use of the client/server architecture with the RIVET server, BOLT client, and CARVE application. The VE in RIVET adapted the base Engineering Test World mission to meet the needs of this type of task, including the terrain, foliage, roadways, and targets (Fig. 25).

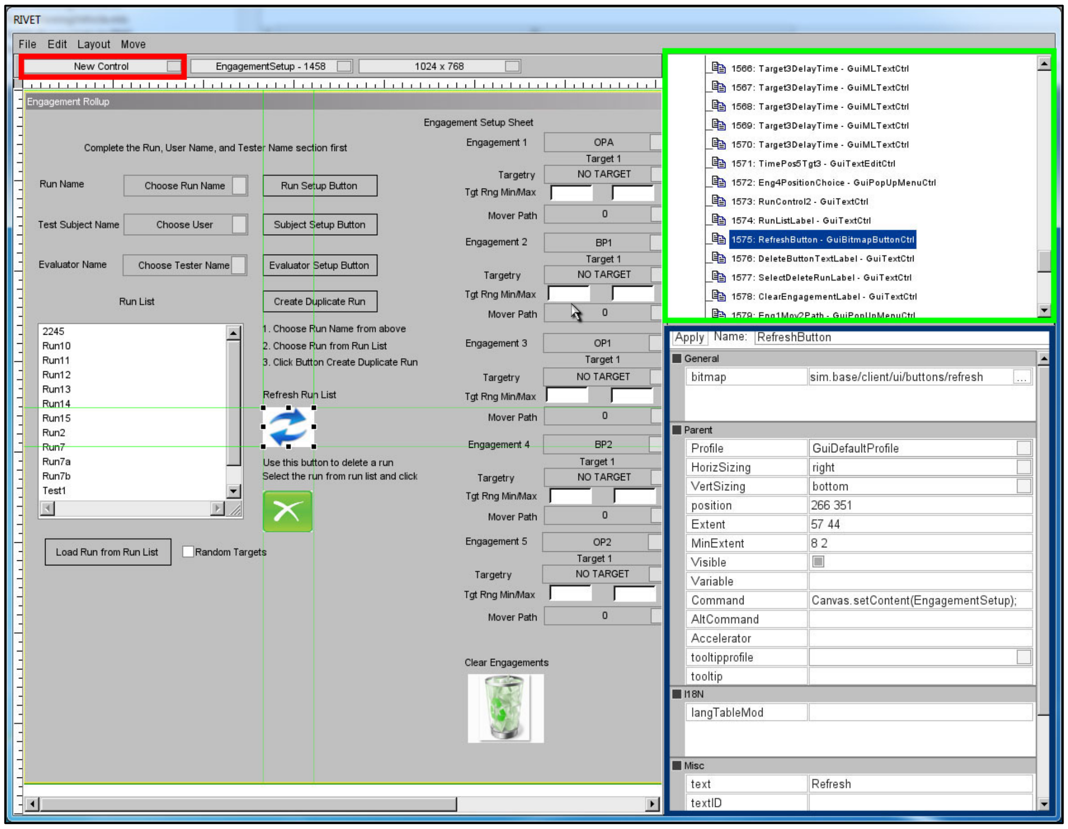


Configuration: Soldier

Side: Neutral

Fig. 25 Gunnery range map

These exercises required additional components to be set up within RIVET, including a number of targets, type of targets, the posture of the vehicle (offensive or defensive), and the position on the range where it will take place. These data are entered into the system using a custom GUI (Fig. 26). When using this GUI, the data entered are added to the database using SQL calls in the script. Once the database is updated with the exercises, the experiment can be conducted. The exercise begins and the timings of all events that happen during that engagement are recorded in the database for use during an AAR.



```
1566: Target3DelayTime - GuiMLTextCtrl
1567: Target3DelayTime - GuiMLTextCtrl
1568: Target3DelayTime - GuiMLTextCtrl
1569: Target3DelayTime - GuiMLTextCtrl
1570: Target3DelayTime - GuiMLTextCtrl
1571: TimePos5Tgt3 - GuiTextEditCtrl
1572: Eng4PositionChoice - GuiPopUpMenuCtrl
1573: RunControl2 - GuiTextCtrl
1574: RunListLabel - GuiTextCtrl
1575: RefreshButton - GuiBitmapButtonCtrl
1576: DeleteButtonTextLabel - GuiTextCtrl
1577: SelectDeleteRunLabel - GuiTextCtrl
1578: ClearEngagementLabel - GuiTextCtrl
1579: Eng1Mov2Path - GuiPopUpMenuCtrl
```

Name: RefreshButton

General	
bitmap	sim.base/client/ui/buttons/refresh
Parent	
Profile	GuiDefaultProfile
HorizSizing	right
VertSizing	bottom
position	266 351
Extent	57 44
MinExtent	8 2
Visible	▨
Variable	
Command	Canvas.setContent(EngagementSetup);
AltCommand	
Accelerator	
tooltipprofile	
tooltip	
I18N	
langTableMod	
Misc	
text	Refresh
textID	

Note: The GUI creation editor gives the programmer all the tools necessary to create these and other interfaces. New controls are found in the upper left drop-down box (outlined in red), and once they are added to the canvas, they show up in the GUI tree view (outlined in green) with their properties accessible in the inspector dialog (outlined in blue).

Fig. 26 GUI editor: example of gunnery user interface GUI setup

5.3 Driverless Vehicle Transport

A third, and most recent, type of mission used for HRI experimentation investigated a specific non-combat operation of driverless vehicles for passenger transit. One example is an experiment motivated by the Autonomous Robotics for Installations and Base Operations (ARIBO) project (Marshall 2014). The purpose of this experiment was to build a virtual US Army post in RIVET in order to assess behavior and performance of a person on board a self-driving vehicle prior to the actual vehicle being developed (Schaefer and Scribner 2015). This design required use of the client/server architecture with the RIVET server, BOLT client, and CARVE application.

The first major need was to build the VE within RIVET (refer to Appendixes A–I). For this design, the ARL Test World was used as the base virtual terrain. The height of the terrain varies as seen from the mountain ranges in the background to the mostly level terrain in the cantonments area. However, this open expanse amidst a mountain range was sufficient for this mission environment. The ground has multiple textures including grass, dirt, and hard-packed black gravel. These were added to the scene with the Terrain Texture editor that allows the creator to add color and dimension to the scene. Then, the medical facilities from a US military installation were used as a frame of reference for developing a road and sidewalk structure, building location, parking lots, and other stationary obstacles (Fig. 27). Many of the static objects included in this new mission were already available within the RIVET libraries. While this means that the VE is not an exact replica of the real environment, it does meet the requirements for the experiment and provides the capability to include some of the key requirements (e.g., vehicle behaviors, transportation times, number of turns) to be able to transition this experimental design to the real world once the prototype robotic vehicle is available for further experimentation.

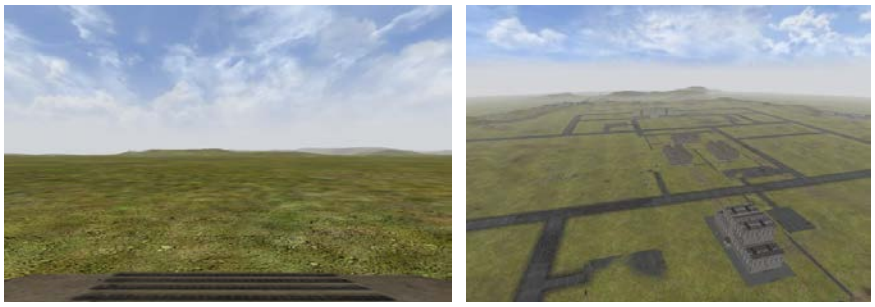
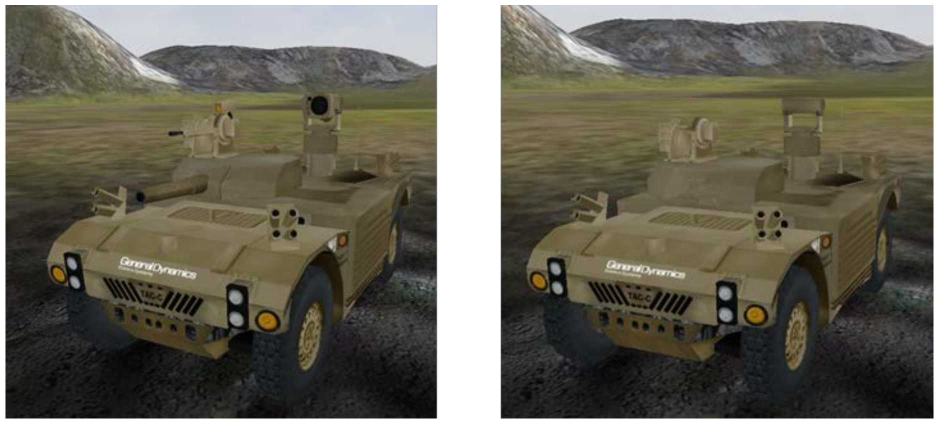


Fig. 27 Virtual environment representing the medical facilities on a US Army post

In the next step after building the VE, the CARVE application was adapted in multiple ways. First, dynamic waypoints were included (refer to Section 4.2.3) to more accurately mirror real-world autonomous behaviors. Second, a new controller, the Logitech G27 wheel and pedals, as well as the engage/disengage buttons, were integrated into CARVE for realistic manual control. Third, an output file (.csv) was created that recorded all interventions in the vehicle's autonomy. Finally, a number of GUIs were developed within RIVET to provide instructions and information to the participant (e.g., notice of other passengers onboard the vehicle, location information, and wrong-way directions).

The final step was to choose a robotic vehicle that was approximately the same size and shape of the real-world prototype vehicle. Even though there are many different robotic vehicles to choose from within the simulation, it is important to match the system to the exercise. A larger vehicle was needed for this experiment, as it has to be able to pick up and discharge passengers. Since details were not available about the final physical design of the real-world prototype vehicle, the tactical autonomous combat chassis (TAC-C) was used, which was previously designed by GDRS as a simulated manned/unmanned platform with the potential for manual or autonomous control (Fig. 28). In addition, the camera position was moved to be located at the "driver" position of the vehicle, so very little of the vehicle was actually seen by the participant. Additional information on this set of experiments is currently available in Schaefer and Scribner (2015) and will be available in additional ARL technical reports.



Note: The TAC-C was used because of its size (height and width), as well as the functionality of the animation sequences. Originally, the vehicle had a turret and other associated weaponry (left). Once the graphics of the turret and other weaponry were removed, the functionality to scan right to left as well as up and down was still there (right). We used this functionality to represent someone turning their head to look around for oncoming traffic or environmental obstacles.

Fig. 28 Customizations to the TAC-C

6. Discussion

This report provides an introduction to developing and advancing RIVET simulation and the associated CARVE application to meet the needs for HRI experimentation. Overall, the capability for customization from the user control interface to the sensor and autonomy integration and on to the integration of a user feedback interface make the RIVET simulation system a valuable resource to understanding the needs of a variety of current and future HRI (Section 5). Further, step-by-step instructions are provided for future researchers to set up and use RIVET (Appendixes A–M). This section provides some remaining lessons learned while updating the RIVET system, as well as additional resource guides.

6.1 Lessons Learned

This section addresses some of the solutions to problem areas found while developing and advancing RIVET and the associated CARVE application.

6.1.1 Problem No. 1: CARVE Does Not Always Load Correctly

If CARVE does not load, gives an error, or shuts down, then first check the connectivity to make sure the CARVE computer is receiving signals from both the RIVET server and the BOLT client computers (Appendix B, Figs. B-1 and B-2). It is good practice to look at the connectivity prior to loading CARVE to check that CARVE is set up to connect to the RIVET and BOLT IP addresses through the Ethernet LAN.

The second check is the RIVET server machine. When using the 3-computer setup (RIVET, BOLT, CARVE), it is vital to choose a vehicle that has a BOLT sensor attachment. Otherwise, the BOLT sensor on the client machine and the CARVE application will not connect. If the correct vehicle was chosen, then check the BOLT client machine. Double-check to see that the name is set to User 1, the name of the vehicle. This name is what allows the BOLT sensor to attach to the vehicle.

If there are still issues connecting to the CARVE application, the final thing to check is the map. CARVE requires the map of the VE from RIVET to function appropriately. The optimal sizing for the map is 1600 × 1600 pixels and less than 2 MB in size. External programs can be used to reduce the size of the .png file (e.g., Adobe Fireworks). Directions for creating a map are provided in Appendix B.

6.1.2 Problem No. 2: CARVE Takes Too Long to Load

CARVE can take upwards of a minute to initially load the application, because it is pulling the mission map from RIVET every time it is initialized. When conducting multiple trials, this time delay can add up to a great deal of wait time. However, to address this issue, we built in a shortcut that will store the map locally following the first initialization of CARVE. Directions are included in Appendix K.

6.1.3 Problem No. 3: Picture Quality Is Granulated or Pixelated

If the picture becomes granulated or pixelated, this may be a sign of a graphics card issue. Building complex VEs, especially those with a number of moving objects, can slow down the system. One solution is to reduce the number of static and dynamic objects in the VE. However, that may not always be possible. A second solution is to use trigger points to create and delete NPC and path-following vehicles at set points in time and space during the mission run (Appendixes E–H). Finally, check the hardware—more specifically, the graphics card. Nvidia GeForce graphics cards were found to be better suited to work with complex VEs within RIVET than the Nvidia Quadro cards. In addition, it may be beneficial to have a graphics card with a separate memory storage (e.g., Nvidia GeForce GTX 780).

If this issue only occurs on the CARVE display, then it may be a packaging issue. CARVE pulls the camera video from the BOLT machine by decompressing the video stream and then enlarging it on the CARVE screen. This process of shrinking and then enlarging the video stream can lead to pixelation. One possible solution is to reduce the screen size on the BOLT machine by clicking on the Configuration button prior to mounting the BOLT sensor (Fig. 29).

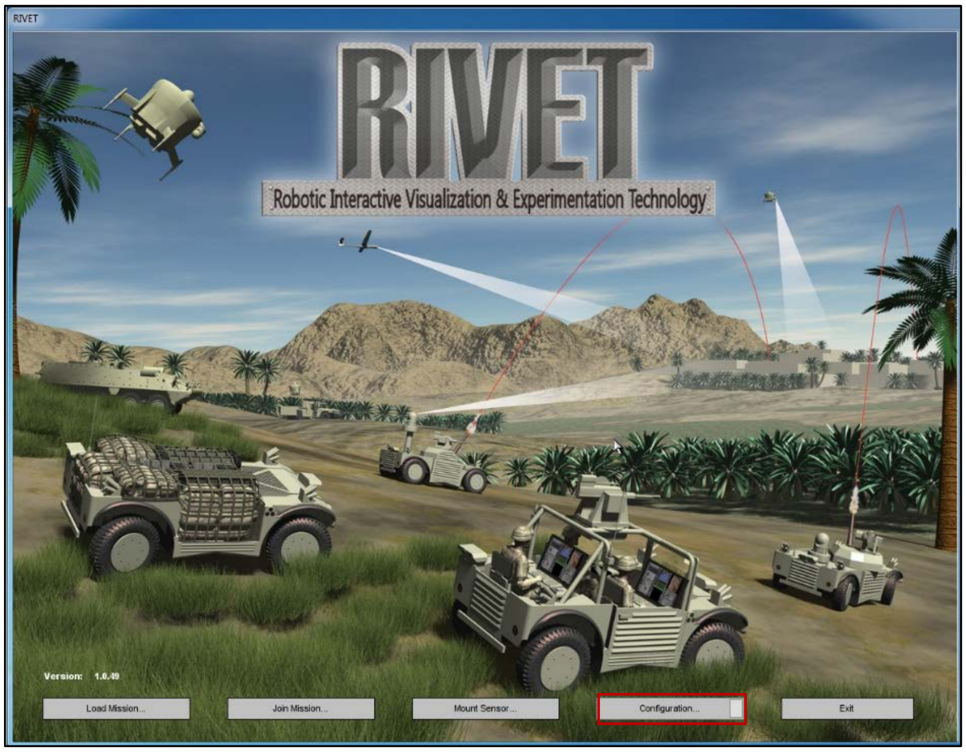


Fig. 29 Location to change the screen size

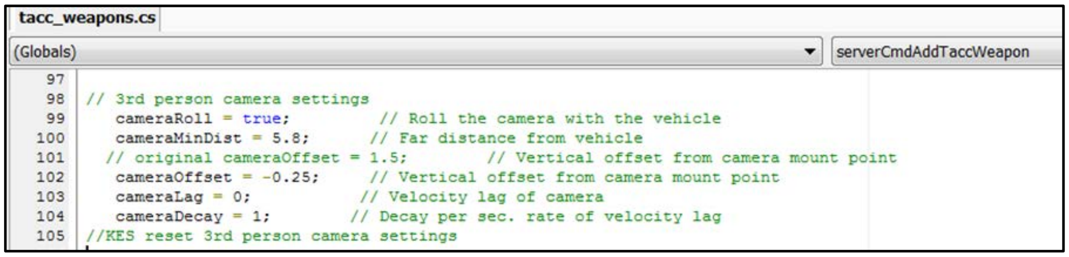
6.1.4 Problem No. 4: Camera Sensor Is Located in the Wrong Place

It is possible that the standard camera sensor location is not in ideally placed to meet the needs of the experiment. For example, if the camera is placed too high or too low, a Soldier avatar may appear to be taller or shorter than an average person appears. Similarly, the height of the camera sensor on a vehicle may give the perspective of driving on roof if placed too high, or display the animations of the wheel wells if placed too low. In addition, the forward or rear placement of the camera sensor can change the perspective of the experiment. For example, with a Soldier avatar, it is possible to change the camera placement to be in front of the Soldier, at eye level of the Soldier, or behind the Soldier.

The solution for changing the placement of the camera is simple, but it requires an understanding of the TorqueScript files. Every player vehicle has an associated TorqueScript file (.cs) that can be located in the following directory: C:\RIVET\sim.base\server. To edit the specific vehicle file (.cs), it will need to be opened as a text file in Notepad, Notepad++, or Torsion.

The following are step-by-step directions on how to adapt the camera sensor placement for the TAC-C.

1. Open the tacc_weapons.cs file (C:\RIVET\sim.base\server).
2. Scroll down or search (Ctrl+f) for //3rd person camera settings (around line 98).
3. To change the camera's distance either forward or backward from the middle of the vehicle, change the cameraMinDist number. The max numbers are +8.0 and –8.0.
4. To change the vertical distance higher or lower, change the cameraOffset with a positive or negative number (Fig. 30).



```
 97
 98  // 3rd person camera settings
 99     cameraRoll = true;          // Roll the camera with the vehicle
100     cameraMinDist = 5.8;       // Far distance from vehicle
101   // original cameraOffset = 1.5;         // Vertical offset from camera mount point
102     cameraOffset = -0.25;      // Vertical offset from camera mount point
103     cameraLag = 0;            // Velocity lag of camera
104     cameraDecay = 1;         // Decay per sec. rate of velocity lag
105  //KES reset 3rd person camera settings
```

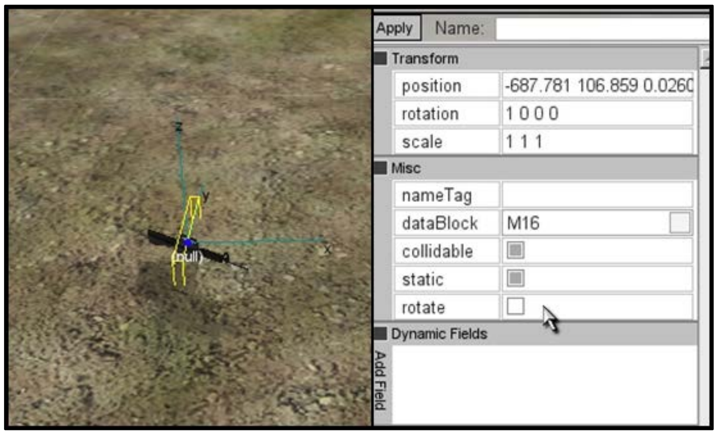
Note: Changing the location of the camera sensor in this file will change the placement of the camera sensor for this vehicle when used in any mission in the future. Therefore, it may be beneficial to add a comment (green text denoted with 2 forward slashed lines) marking changes made to the TorqueScript file.

Fig. 30 TorqueScript file in Torsion for adapting the camera sensor placement for the TAC-C vehicle

6.1.5 Problem No. 5: Need to Customize Pre-loaded Animation Sequences

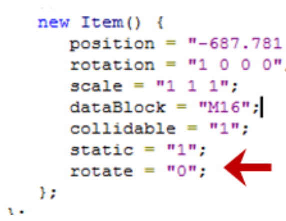
The current version of RIVET is designed to initialize some set animation sequences upon inclusion in a mission environment. For example, in the experiment on target detection (Section 5.1.2), upon placing the M16 rifles within the VE, they began to spin, as is common in game design. This animation sequence may or may not be ideal for a specific experiment; however, this animation sequence has also been programmed to be customizable. In other words, it is possible to stop the rifles from spinning in circles. Although this specific example refers to rifles that spin, other animation sequences might be customized in a similar manner. Some brief directions to customize are as follows:

1. In RIVET, press F11 to open the World Editor.
2. Press F4 (or use the Window menu) to open the World Editor Creator. Select and add the M16 rifle from the menu on the right side of the screen.
3. To add the M16 rifle, select “shapes”, “weapons”, and “M16”. A spinning rifle should be visible in the VE.
4. Click on the M16. Once it has a yellow box, then press F3 (or use the Windows menu) to open the World Editor Inspector to customize the M16 rifle.
5. To stop the rifle from spinning, deselect the Rotate button and click apply (Fig. 31).
6. The rifle will continue spinning. Go to File, click Save, and then logout of RIVET completely. After logging back in to the Mission file, the rifle should no longer be spinning.
7. Another way to ensure that it accepted the changes is by looking at the TorqueScript file. To do this, open the Mission File (.mis) by going to the directory C:\RIVET\sim.base\data\usermissions\ and select the correct Mission file. At the bottom of the file, the new item should appear. In the example shown in Fig. 32, the dataBlock is an M16 and the rotate is set to “0” or false (marked with a red arrow).



position	-687.781 106.859 0.0260
rotation	1 0 0 0
scale	1 1 1
nameTag	
dataBlock	M16
collidable	▣
static	▣
rotate	☐

Fig. 31 Example of adapting animations: spinning M16



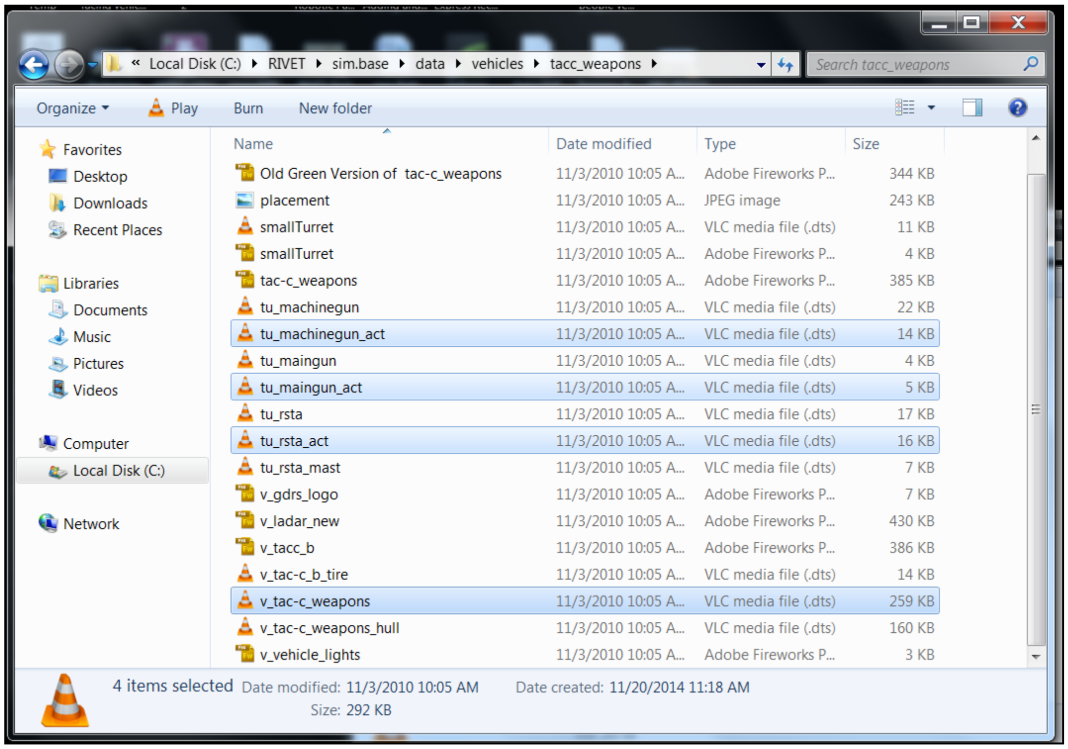
```
new Item() {
   position = "-687.781 106.859 0.0260671";
   rotation = "1 0 0 0";
   scale = "1 1 1";
   dataBlock = "M16";
   collidable = "1";
   static = "1";
   rotate = "0";
};
```

Fig. 32 Example code for customizing rotation animations on the M16 rifle

6.1.6 Problem No. 6: Removing Add-ons from Pre-loaded Graphics and Animations

The current design of RIVET includes a number of accessible graphics and animations. However, in their original state it may not be possible to directly integrate those graphics into a new mission. For example, in the experimental design using the TAC-C vehicle in place of a future self-driving vehicle for Soldier transport in a noncombat environment (Section 5.3), even after relocating the camera sensor, it was still possible to see the weapons systems. Since the design of this vehicle was built in multiple pieces, it was possible to simply remove the graphics (i.e., the weapons systems) while maintaining the functionality of a turret, from which the camera can move through the BOLT interface. Some quick instructions for removing the weapons systems graphics from the TAC-C vehicle are as follows:

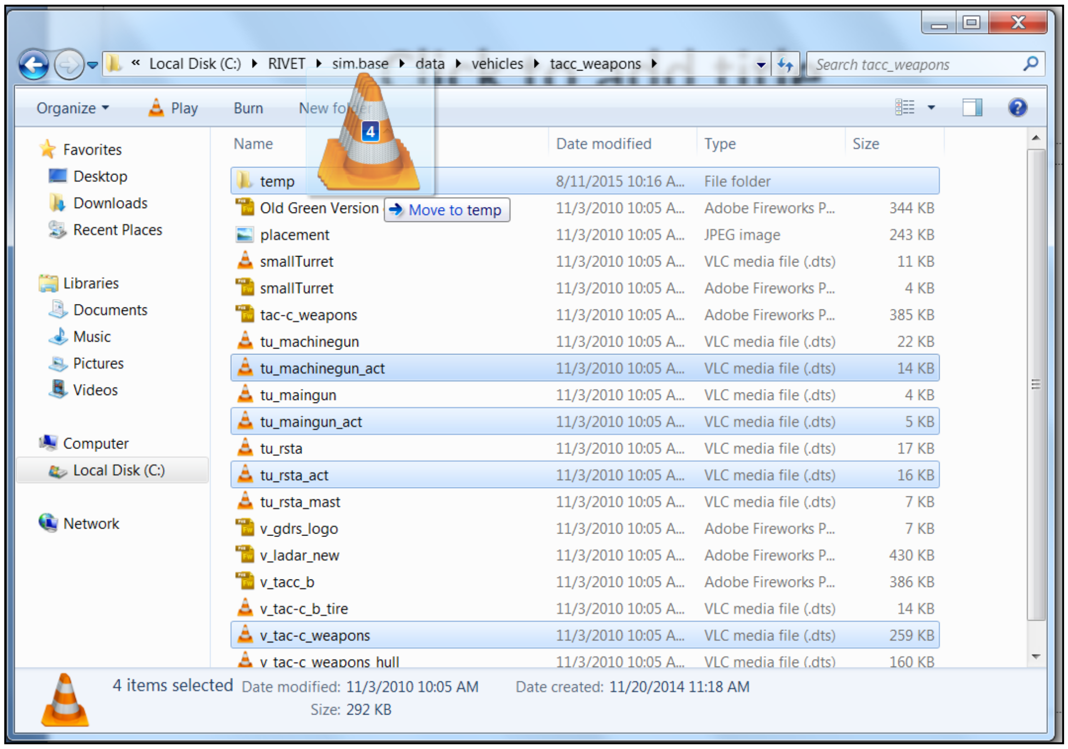
1. Open the directory: C:\RIVET\sim.base.data\vehicles\tacc_weapons.
2. Select the following 4 .dts media files: tu_machinegun_act; tu_maingun_act; tu_rsta_act; and v_tac-c_weapons (Fig. 33).
3. Do not delete these files. Create a new folder in this directory called temp, and place the files in this folder (Fig. 34) to allow the files to be returned to the directory at a later time.



Name	Date modified	Type	Size
Old Green Version of tac-c_weapons	11/3/2010 10:05 A...	Adobe Fireworks P...	344 KB
placement	11/3/2010 10:05 A...	JPEG image	243 KB
smallTurret	11/3/2010 10:05 A...	VLC media file (.dts)	11 KB
smallTurret	11/3/2010 10:05 A...	Adobe Fireworks P...	4 KB
tac-c_weapons	11/3/2010 10:05 A...	Adobe Fireworks P...	385 KB
tu_machinegun	11/3/2010 10:05 A...	VLC media file (.dts)	22 KB
tu_machinegun_act	11/3/2010 10:05 A...	VLC media file (.dts)	14 KB
tu_maingun	11/3/2010 10:05 A...	VLC media file (.dts)	4 KB
tu_maingun_act	11/3/2010 10:05 A...	VLC media file (.dts)	5 KB
tu_rsta	11/3/2010 10:05 A...	VLC media file (.dts)	17 KB
tu_rsta_act	11/3/2010 10:05 A...	VLC media file (.dts)	16 KB
tu_rsta_mast	11/3/2010 10:05 A...	VLC media file (.dts)	7 KB
v_gdrs_logo	11/3/2010 10:05 A...	Adobe Fireworks P...	7 KB
v_ladar_new	11/3/2010 10:05 A...	Adobe Fireworks P...	430 KB
v_tacc_b	11/3/2010 10:05 A...	Adobe Fireworks P...	386 KB
v_tac-c_b_tire	11/3/2010 10:05 A...	VLC media file (.dts)	14 KB
v_tac-c_weapons	11/3/2010 10:05 A...	VLC media file (.dts)	259 KB
v_tac-c_weapons_hull	11/3/2010 10:05 A...	VLC media file (.dts)	160 KB
v_vehicle_lights	11/3/2010 10:05 A...	Adobe Fireworks P...	3 KB

4 items selected Date modified: 11/3/2010 10:05 AM Date created: 11/20/2014 11:18 AM
Size: 292 KB

Fig. 33 Directory location of files to remove weapons systems from the TAC-C



Name	Date modified	Type	Size
temp	8/11/2015 10:16 A...	File folder	
Old Green Version	11/3/2010 10:05 A...	Adobe Fireworks P...	344 KB
placement	11/3/2010 10:05 A...	JPEG image	243 KB
smallTurret	11/3/2010 10:05 A...	VLC media file (.dts)	11 KB
smallTurret	11/3/2010 10:05 A...	Adobe Fireworks P...	4 KB
tac-c_weapons	11/3/2010 10:05 A...	Adobe Fireworks P...	385 KB
tu_machinegun	11/3/2010 10:05 A...	VLC media file (.dts)	22 KB
tu_machinegun_act	11/3/2010 10:05 A...	VLC media file (.dts)	14 KB
tu_maingun	11/3/2010 10:05 A...	VLC media file (.dts)	4 KB
tu_maingun_act	11/3/2010 10:05 A...	VLC media file (.dts)	5 KB
tu_rsta	11/3/2010 10:05 A...	VLC media file (.dts)	17 KB
tu_rsta_act	11/3/2010 10:05 A...	VLC media file (.dts)	16 KB
tu_rsta_mast	11/3/2010 10:05 A...	VLC media file (.dts)	7 KB
v_gdrs_logo	11/3/2010 10:05 A...	Adobe Fireworks P...	7 KB
v_ladar_new	11/3/2010 10:05 A...	Adobe Fireworks P...	430 KB
v_tacc_b	11/3/2010 10:05 A...	Adobe Fireworks P...	386 KB
v_tac-c_b_tire	11/3/2010 10:05 A...	VLC media file (.dts)	14 KB
v_tac-c_weapons	11/3/2010 10:05 A...	VLC media file (.dts)	259 KB
v_tac-c_weapons_hull	11/3/2010 10:05 A...	VLC media file (.dts)	160 KB

4 items selected Date modified: 11/3/2010 10:05 AM Date created: 11/20/2014 11:18 AM
Size: 292 KB

Fig. 34 Creation of a temporary folder

6.2 Support and Documentation

6.2.1 RIVET Developer's Guide

Perhaps one of the most useful and relevant sources to understanding the makeup and architecture of RIVET is the *RIVET Developer's Guide* (General Dynamics Robotic Systems 2010). This document is included in the software and provides an introduction to the TGE (Chapter 1), system requirements (Chapter 2), a high-level overview of RIVET (Chapter 3), instructions for creating objects (Chapters 4–7), instructions for world building and introduction to the editors (Chapter 8), introduction to TorqueScript (Chapter 9), entity AI (Chapter 10), dataset creation (Chapter 11), and HITL support (Chapter 12–13). However, this resource is a proprietary document that may also be difficult to understand without additional background in the TGE, game development, or computer programming.

6.2.2 GarageGames.com Online Forums and Documentation

Torque Developer Network (tdn.garagegames.com) is the official reference site for coders, artists, or anyone working with Torque. This site requires a login and is currently in its beta version. It has a number of tutorials and documentation for a wide range of topics. Since all materials have been written collaboratively by Torque developers, it can sometimes be difficult to understand by a novice Torque user as it has varying levels of descriptions and explanations. Also, some of the step-by-step directions reference editors that are not available in the RIVET v.1.0. However, the documentation often provides examples of TorqueScript files that can be leveraged to meet a user's needs and is constantly updated with new information. In addition, a number of resources are also freely available (http://docs.garagegames.com/tge/official/).

6.2.3 The Game Programmer's Guide to Torque

The Game Programmer's Guide to Torque (Maurina 2006) is a GarageGames book that provides a cogent, base-level introduction to Torque from start (opening the TGE for the first time) to finish (putting it all together). The core substance of the book focuses on the game elements including classes, objects, and even an introductory glance into scripting and game interfaces. While this book provides a number of step-by-step examples, the design and setup of RIVET extends beyond the standard layout of the TGE. In addition, it may not go into the level of detail needed for creating HRI missions but overall is a valuable resource.

6.2.4 Scripting

While this report does not provide a lot of detail about TorqueScript, understanding the "what" and the "how" can add customization capabilities to the VE. To begin, TorqueScript is a programming language very similar to C++. Having an introductory understanding of programming languages can help immensely, especially when reading different resources. This report provides an overview of some additional resources to scripting. A general introduction to TorqueScript can be found by reviewing the GarageGames online documentation. This information is available online at http://docs.garagegames.com/tgea/official/content /documentation/Scripting%20Reference/Introduction/TorqueScript.html. For a more in-depth review, the book *Advanced 3D Game Programming All in One* (Finney 2005) comes with a CD and provides good explanations of datablocks, vectors, and matrices, along with an entire chapter on TorqueScript. In addition, the book *3D Game Programming All in One* (Finney 2007) provides an introduction to the entire process of creating a game within Torque, including building the models, creating animations, server-client architectures, creation of GUIs, and even testing.

6.2.5 Animations

Chapter 7 of the *RIVET Developers Guide* (General Dynamics Robotic Systems 2010) provides the most in-depth instruction as to creating animated models. There are multiple choices for modeling and rendering applications such as Blender, Maya, and 3ds Max. The instructions in the Guide are based on 3ds Max by Autodesk.

6.2.6 Other Torque Resources

Finally, other online resources may be helpful in providing guidance and instruction while developing specific missions, including tutorials by Dr David J Sushil: *A Torque Game Engine Primer* (2008), *Simple AI in the Torque Game Engine* (2008), *Creating Doors in Torque* (2007), *Animating Characters in Torque* (2007), *Sound Effects in Torque* (2007), and *Controlling Events with Triggers in Torque* (2007). These resources, as well as a number of examples were written as instructional guides for game development students. These step-by-step guides often provide more detail than programming books or online forums. In addition, they are freely downloadable .pdf files that can be found online at http://www.davidjsushil.com /index.php?action=tutorials.

7. Conclusions

Using a game engine (such as RIVET) to create a robotic platform application, while including a separate interface for human interaction, provides a division between the software applications. This separation allows software engineers to make changes to the algorithms in perception, intelligence, and mobility while leaving the HRI untouched. Therefore, development of each part may continue while still allowing for testing at each step. The game engine allows for reproducibility during testing to capture the user interactions with the robot. There is no longer a need to wait for the final product to roll off the assembly line and into the hands of Soldiers before conducting experimentation to explore human interaction questions. Instead, integrating HRI experimentation early on can help drive design choices during the stages of development.

8. References

Basdogan C, De S, Kim J, Muniyandi M, Kim H, Srinivasan MA. Haptics in minimally invasive surgical simulation and training. Computer Graphics and Applications, IEEE. 2004;24(2):56–64.

Bethel CL, Salomon K, Murphy RR, Burke JL. Survey of psychophysiology measurements applied to human-robot interaction. In: Robot and human interactive communication, 2007. RO-MAN 2007. The 16th IEEE International Symposium; 2007 Aug 26–29; Jeju, South Korea. New York (NY): IEEE; c2007. p. 732–737. doi: 10.1109/ROMAN.2007.4415182.

Bicho E, Louro L, Erlhagen W. Integrating verbal and nonverbal communication in a dynamic neural field architecture for human-robot interaction. Frontiers in Neurorobotics. 2010;4(5). doi: 10.3389/fnbot.2010.00005.

Bodt BA, Childers MA, Hill SG, Camden RS, Gonzalez JP, Dean RM, Dodson WF, Kreafle G, LaCaze A, Sapronov L. Unmanned ground vehicle two-level planning technology assessment. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2010. Report No.: ARL-TR-5331.

Boehm BW. A spiral model of software development and enhancement. Computer. 1988;21(5):61–72.

Cassenti DN, Kelley TD, Avery E, Yagoda RE. Location label speech options improve robot operator performance. In: Proceedings of the Human Factors and Ergonomics Society. Thousand Oaks (CA): SAGE Publications; 2011;55(1): 439–443. doi: 10.1177/1071181311551090.

Chen JYC, Barnes MJ. Human-agent teaming for multirobot control: a review of human factors issues. IEEE Transactions on Human-Machine Systems. 2014;44(1):13–29. doi:10.1109/THMS.2013.2293535.

Chen JYC, Barnes MJ, Qu Z, Snyder MG. RoboLeader: an intelligent agent for enhancing supervisory control of multiple robots. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2010. Report No.: ARL-TR-5239.

Chen JYC, Terrence PI. Effects of imperfect automation and individual differences on concurrent performance of military and robotics tasks in a simulated multitasking environment. Ergonomics. 2009;52:907–920.

Dautenhahn K. Socially intelligent robots: dimensions of human–robot interaction. Philosophical Transactions of the Royal Society B: Biological Sciences. 2007;362(1480):679–704.

Davies B. A review of robotics in surgery. Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine. 2000;214:129–140. doi: 10.1243/0954411001535309.

Dean RMS, DiBerardino CA. Robotics Collaborative Technology Alliance: An open architecture approach to integrated research. In: Suresh R, editor. Proceedings of SPIE, Open Architecture/Open Business Model of Net-Centric Systems and Defense Transformation; 2014; Baltimore (MD): International Society for Optics and Photonomics.

Finney KC. Advanced 3D game programming all in one. Boston (MA): Thomson Course Technology; 2005.

Finney KC. 3D game programming all in one. 2nd ed. Boston (MA): Thomson Course Technology; 2007.

General Dynamics Robotic Systems. RIVET developer's guide. (included in software installer package in Docs folder). Westminster (MD): General Dynamics Robotics Systems; 2010.

Gerkey BP, Vaughan RT, Howard A. The player/stage project: tools for multi-robot and distributed sensor systems. In: Proceedings of the 11th International Conference on Advanced Robotics. ICAR 2003; 2003 Jun 30–Jul 3; Coimbra, Portugal. p. 317–323.

Gonzalez JP, Dodson W, Dean R, Kreafle G, Lacaze A, Sapronov L, Childers M. Using RIVET for parametric analysis of robotic systems. Proceedings of the 2009 Ground Vehicle Systems Engineering and Technology Symposium; 2009 Aug 18–20; Troy, MI.

Goodrich MA, Schultz AC. Human-robot interaction: a survey. Foundations and trends in human-computer interaction. 2007;1(3):203–275. doi: 10.1561/1100000005.

Groom V. What's the best role for robot? Cybernetic models of existing and proposed human-robot interaction structures. In: Filipe J, Andrade-Cetto J, Ferrier J, editors. Proceedings of the Fifth International Conference on Informatics in Control, Automation, and Robotics; ICINCO-RA 2008; 2008 May 11–15; Funchal, Madeira, Portugal. Funchal (Portugal): INSTICC Press; c2008. p. 323–328.

Her MG, Hsu KS. Design and analysis of haptic direct drive robot for virtual reality. Journal of Information and Optimization Sciences. 2001;22(3):441–461.

Hughes S, Lewis M. Robotic camera control for remote exploration. In: Proceedings of ACM CHI 2004 Conference on Human Factors in Computing Systems; 2004 Apr 24–29; Vienna, Austria. New York (NY): ACM; c2004. p. 511–517.

Kunkler K. The role of medical simulation: an overview. The International Journal of Medical Robotics and Computer Assisted Surgery. 2006;2:203–210. doi: 10.1002/rcs.101.

Lee JD, See KA. Trust in automation: designing for appropriate reliance. Human Factors: the Journal of the Human Factors and Ergonomics Society. 2004;46(1):50–80. doi:10.1518/hfes.46.1.50_30392.

Lewis M, Wang J, Hughes S. USARSim: simulation for the study of human-robot interaction. Journal of Cognitive Engineering and Decision Making. 2007;1(1):98–120.

Loper ML. The modeling and simulation life cycle process. In: Loper M, editor. Modeling and simulation in systems engineering life cycle: core concepts and accompanying lectures. London (England): Springer-Verlag; 2015. p. 17–27.

Marshall P. Army tests driverless vehicles in ‘living lab’. Vienna (VA): GCN Technology, Tools and Tactics for Public Sector IT. 2014 Jul 16 [accessed 2014 Sep 8]. http://gcn.com/articles/2014/07/16/aribo-armytardec.aspx.

Maurina EF III. The game programmer’s guide to torque: under the hood of the Torque game engine. Wellesley (MA): GarageGames, Inc; 2006.

Michel O. Webots: symbiosis between virtual and real mobile robots. In: Virtual Worlds. Berlin Heidelberg (Germany): Springer-Verlag; 1998 Jan; p. 254–263.

Murphy RR, Schreckenghost D. Survey of metrics for human-robot interaction. In: Proceedings of the 8th ACM/IEEE Human-Robot Interaction Conference; HRI 2013; 2013 Mar 3–6; Tokyo, Japan; Piscataway (NJ): IEEE; c2013. p. 197–198.

Nielsen C, Ricks B, Goodrich M, Bruemmer D, Few D, Walton M. Snapshots for semantic maps. In: Proceedings of the 2004 IEEE International Conference on Systems, Man, and Cybernetics; 2004 Oct 10–13; The Hague, Netherlands. Piscataway (NJ): IEEE; c2004. p. 2853–2858.

Nourbakhsh I, Sycara K, Koes M, Yong M, Lewis M, Burion S. Human-robot teaming for search and rescue. Pervasive Computing, IEEE. 2005;4(1):72–79.

Olsen D Jr, Wood S. Fan-out: measuring human control of multiple robots. In Proceedings of ACM CHI 2004 Conference on Human Factors in Computing Systems; 2004 Apr 24–29; Vienna, Austria. New York (NY): ACM; c2004. p. 231–238.

Parasuraman R, Galster S, Miller C. Human control of multiple robots in the RoboFlag simulation environment. In: IEEE International Conference on Systems, Man and Cybernetics, 2003.; 2003 Oct 5–8; Washington, DC. Piscataway (NJ): IEEE; c2003. Vol. 4, p. 3232–3237.

Parasuraman R, Riley V. Humans and automation: use, misuse, disuse, abuse. Human Factors: the Journal of the Human Factors and Ergonomics Society. 1997;39(2):230–253. doi: 10.1518/001872097778543886.

Phillips E, Ososky S, Grove J, Jentsch F. From tools to teammates toward the development of appropriate mental models for intelligent robots. In: Proceedings of the Human Factors and Ergonomics Society Annual Meeting; 2011 Sep 19–23; Las Vegas, NV. Thousand Oaks (CA): SAGE Publications; 2011:55(1):1491–1495.

Preece J, Sharp H, Rogers Y. Interaction design: beyond human-computer interaction. West Sussex (UK): John Wiley & Sons Ltd; 2007.

Rickheit G, Wachsmuth I. Alignment in communication. Künstliche Intelligenz. 2008;22(2):62–65.

Sanders TL, Llorens N, Billings DR, Schaefer KE, Hancock PA, Driskell L, Long T. Physiological and behavioral cues: identifying their potential value in measuring human-robot trust. Poster presented at: Division 21, American Psychological Association; 2012 Aug; Orlando, FL.

Sanders TL, Wixon T, Schafer KE, Chen JY, Hancock PA. The influence of modality and transparency on trust in human-robot interaction. In: Proceedings of the 4th Annual International Inter-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA); 2014 Mar 3–6; San Antonio, TX. Piscataway (NJ): IEEE; c2014. p. 156–159.

Schaefer KE. Measuring trust in human robot interactions: development of the trust perception scale – HRI. In: Lawless W, Mittu R, Wagner A, Sofge D, editors. The intersection of robust intelligence (RI) and trust in autonomous systems; New York (NY): Springer; 2016. p. 191–219.

Schaefer KE. The perception and measurement of human-robot trust [doctoral dissertation]. [Orlando (FL)]: University of Central Florida; 2013.

Schaefer KE, Scribner DN. Individual differences, trust, and vehicle autonomy: a pilot study. In: Proceedings of the Human Factors and Ergonomics Society; 2015 Oct 26–30; Los Angeles, CA. Thousand Oaks (CA): SAGE Publications; c2015.

Schafer KE, Sanders T, Kessler TA, Dunfee M, Wild T, Hancock PA. Fidelity & validity in robotic simulation. In: Proceedings of the 5th Annual International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA); 2015 Mar 9–12; Orlando, FL. Piscataway (NJ): IEEE; c2015. p. 113–117. doi: 10.1109/COGSIMA.2015.7108184.

Sheridan TB. Human supervisory control of robot systems. Robotics and Automation. 1986;3:808–812. doi: 10.1109/ROBOT.1986.1087506.

Steinfeld A, Fong T, Kaber D, Lewis M, Scholtz J, Schultz A, Goodrich M. Common metrics for human-robot interaction. In: Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-Robot Interaction; 2006 Mar 2–3; Salt Lake City, UT. New York (NY): ACM; p. 33–40.

Sushil DJ. A Torque Game engine primer. Orlando (FL): Sushil DJ; 2008 Apr 14 [accessed 2015 Nov 25]. Notes (http://www.davidjsushil.com/primer.pdf).

Sushil DJ. Simple AI in the Torque Game engine. Orlando (FL): Sushil DJ; 2008 Apr 14 [accessed 2015 Nov 25].
Notes (http://www.davidjsushil.com/tutorials/simpleai.pdf).

Sushil DJ. Creating doors in Torque. Orlando (FL): Sushil DJ; 2007 Jun 26 [accessed 2015 Nov 25].
Notes (http://www.davidjsushil.com/tutorials/doors.pdf).

Sushil DJ. Animating characters in Torque. Orlando (FL): Sushil DJ; 2007 Jun 16 [accessed 2015 Nov 25].
Notes (http://www.davidjsushil.com/tutorials/doors.pdf).

Sushil DJ. Sound effects in Torque. Orlando (FL): Sushil DJ; 2007 Jun 9 [accessed 2015 Nov 25].
Notes (http://www.davidjsushil.com/tutorials/playingsounds.pdf).

Sushil DJ. Controlling events with triggers in Torque. Orlando (FL): Sushil DJ; 2007 Apr 23 [accessed 2015 Nov 25].
Notes (http://www.davidjsushil.com/tutorials/playingsounds.pdf).

Sycara K, Lewis M. Experiments in implicit control. In: Notes of the IJCAI 2003 Workshop on Mixed-Initiative Intelligent Systems; 2003 Aug 9; Acapulco, Mexico [accessed 2015 Nov 4] http://www.researchgate.net/profile /Katia_Sycara2/publication/246140754_Experiments _in_Implicit_Control/links/53d7c9740cf2631430bfc378.pdf.

US Army Research Laboratory Robotics Collaborative Technology Alliance (RCTA): FY 2012 annual program plan. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2012 Mar. [accessed 2015 Nov 3] http://www.arl.army.mil/www/pages/392/RCTA_FY12_APP.pdf.

Weiss A, Bernhaupt R, Lankes M, Tscheligi M. The USUS evaluation framework for human-robot interaction. In: Proceedings of the AISB Symposium on New Frontiers in Human-Robot Interaction; 2009 Apr 6–9; Edinburgh, Scotland. Hove (UK): AISB; c2009. Vol. 4, p. 11–26.

INTENTIONALLY LEFT BLANK.

Appendix A. RIVET Menus

Server Computer: Main User. The first button on the main menu is Load Mission (Fig. A-1).

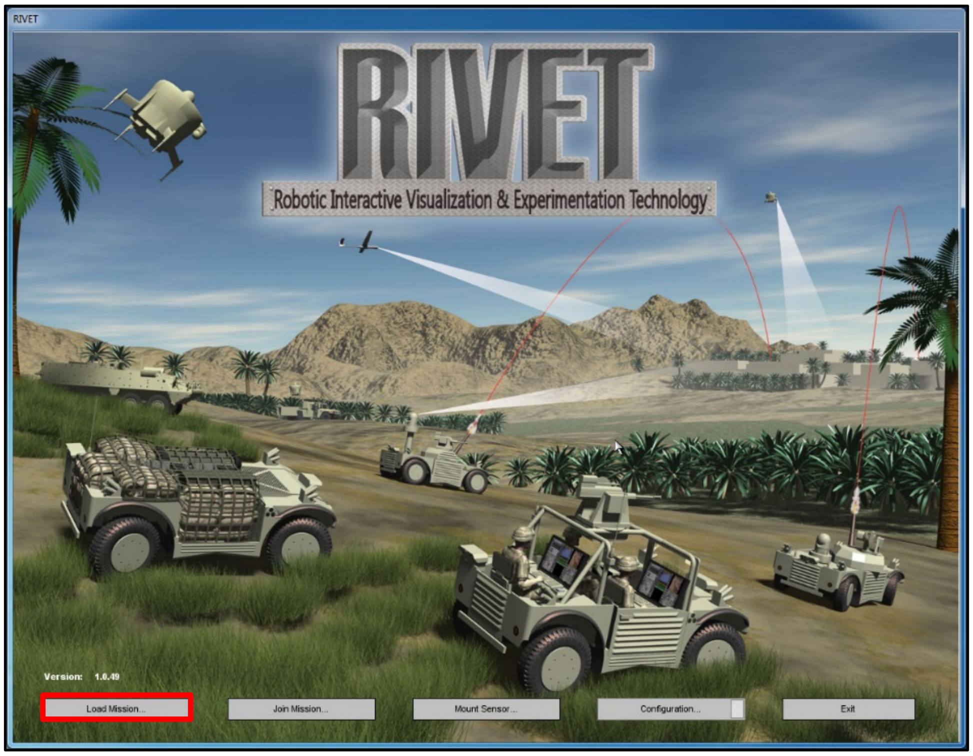
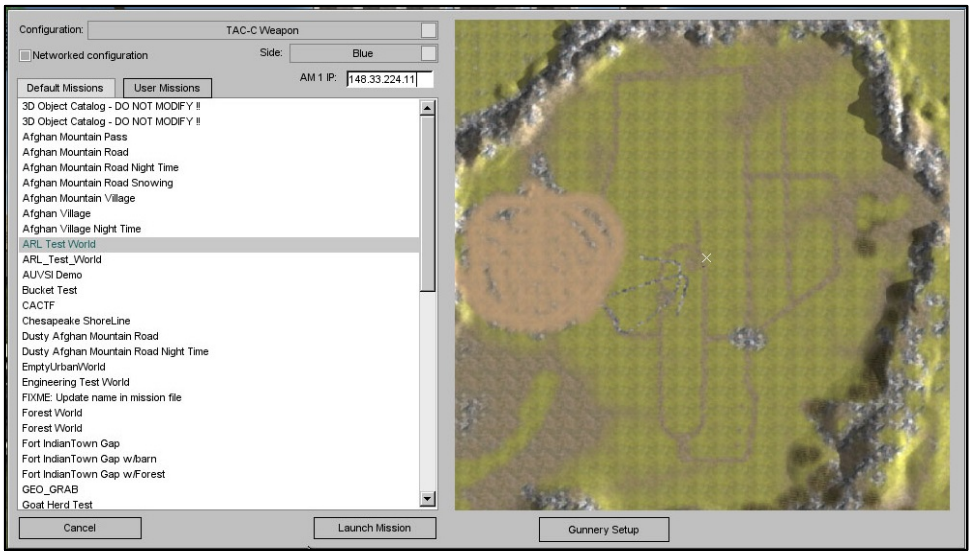


Fig. A-1 RIVET main menu GUI for main user

It leads the user to the next GUI where the user can choose the mission, vehicle, and whether it is friendly or enemy (Fig. A-2). The image on the right is the overhead map of the mission area.



Configuration: TAC-C Weapon

Networked configuration

Side: Blue

AM 1 IP: 148.33.224.11

Default Missions | User Missions

3D Object Catalog - DO NOT MODIFY !!
3D Object Catalog - DO NOT MODIFY !!
Afghan Mountain Pass
Afghan Mountain Road
Afghan Mountain Road Night Time
Afghan Mountain Road Snowing
Afghan Mountain Village
Afghan Village
Afghan Village Night Time
ARL Test World
ARL_Test_World
AUVSI Demo
Bucket Test
CACTF
Chesapeake ShoreLine
Dusty Afghan Mountain Road
Dusty Afghan Mountain Road Night Time
EmptyUrbanWorld
Engineering Test World
FIXME: Update name in mission file
Forest World
Forest World
Fort IndianTown Gap
Fort IndianTown Gap w/barn
Fort IndianTown Gap w/Forest
GEO_GRAB
Goat Herd Test

Fig. A-2 RIVET load menu GUI for main user

Client Computer: Add Additional Players. Up to 64 players can join a mission (Fig. A-3). A player can be a person (e.g., Soldier), vehicle (e.g., Humvee), robot (e.g., Talon), or a sensor attached to the main character (from the Server computer).

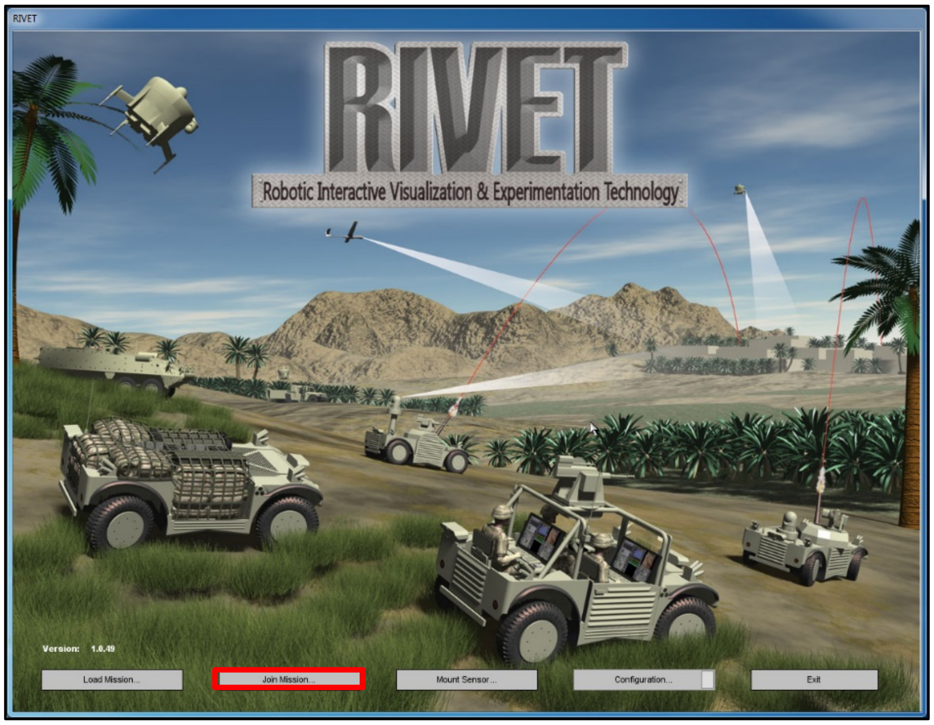
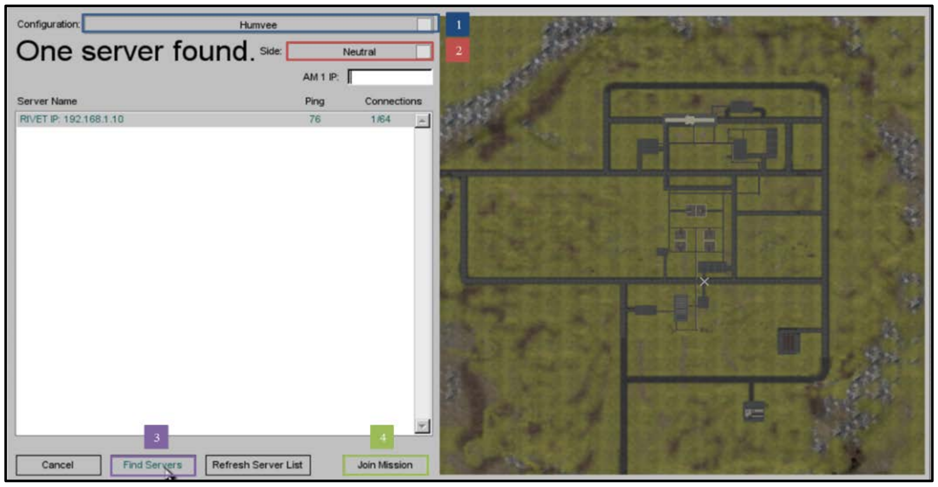


Fig. A-3 RIVET main menu GUI for client computer

After selecting Join Mission, a new window (Fig. A-4) will open that will allow selection of a player character or vehicle [1] and side (i.e., red, blue, neutral) [2]. To connect to the correct Mission, click Find Servers [3] and select the Server Name. To join the mission, select Join Mission [4].


Configuration: Humvee

One server found.

Side: Neutral

AM 1 IP:

Server Name	Ping	Connections
RIVET IP: 192.168.1.10	76	1/64

Fig. A-4 RIVET join mission GUI for client computer

INTENTIONALLY LEFT BLANK.

Appendix B. CARVE Menus

CARVE Main Menu. The Control of Autonomous Robotic Vehicle Experiments (CARVE) Main Menu (Fig. B-1) leads the user to a series of submenus located in the lower-right-hand corner of the screen. These include the Connection button (bottom right), settings submenu (bottom left), user interfaces (top left), and vehicle operations submenu (top right). These 4 icons are always available to the user and are accessed by a mouse interface.

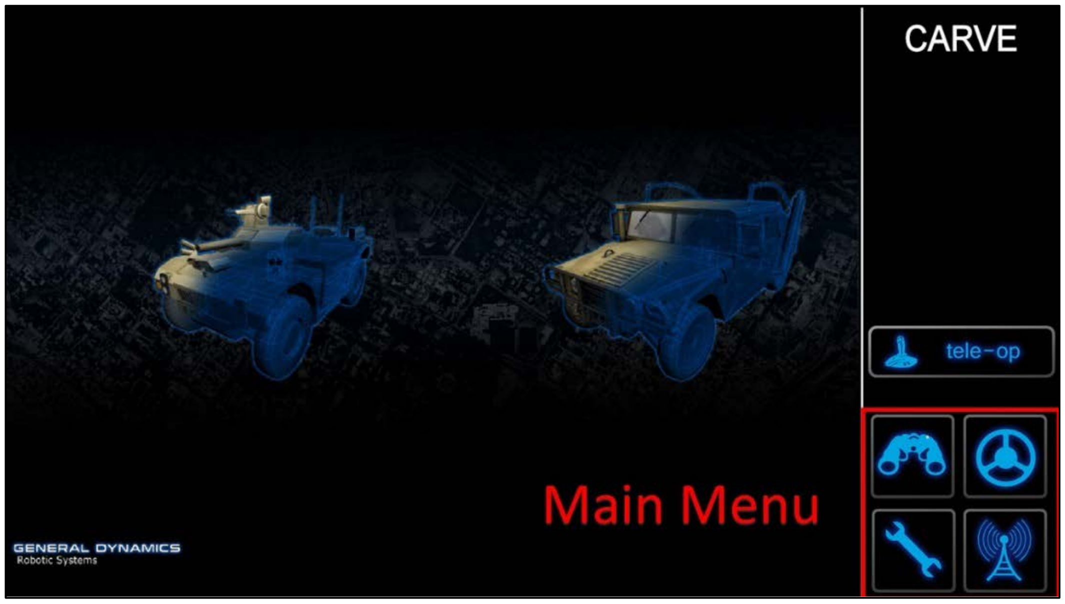


Fig. B-1 CARVE Main Menu GUI

Settings submenu. The settings submenu has a few options for customization of the connectivity, video, networking, controller choice, and exiting the simulation (Fig. B-2).

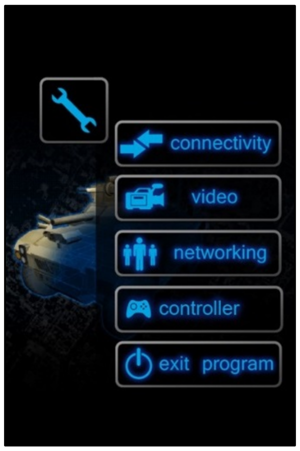


Fig. B-2 Settings submenu

This submenu allows the experimenter to check the setup of the experimental space. This includes being able to set up the connectivity of the sensor and vehicle by entering internet protocol (IP) addresses and ports. Connectivity is needed to connect CARVE to RIVET. The video and networking buttons were designed for later versions of CARVE so that the video and bandwidth can be degraded to represent poor communication situations. The controller button allows the experimenter to set the device that will be used in the mission (e.g., joystick, steering wheel). Finally, to exit the program is self-explanatory.

User Interfaces submenu. The User Interfaces submenu (Fig. B-3) provides the experimenter the capability to set up a number of additional functionalities for the participant.

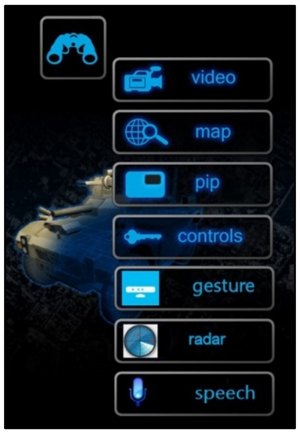


Fig. B-3 User interfaces submenu

The Video button allows the experimenter to set the main screen to show the video stream. The Map button sets the main screen to show the overhead map of the area. The picture-in-picture (PIP) option displays either the map or video opposite to what is displayed on the main screen in a smaller screen in the upper-right-hand corner of the video feed. This gives the user additional situation awareness during experiments. The Controls button provides an on-screen option for controlling the robot or vehicle. While the Gesture and Speech buttons are used to integrate an Xbox Kinect sensor into an experiment if naturalistic communication is required.

Vehicle Operations submenu. The Vehicle Operations submenu (Fig. B-4) allows the user to change the posture in which the vehicle is used.

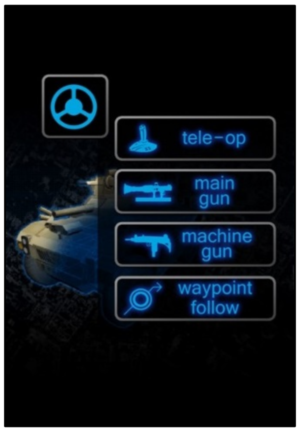


Fig. B-4 Vehicle Operations submenu

Tele-op (which stands for teleoperation) allows the user to drive the vehicle with the chosen controller. These include the game pad, joystick, or racing wheel, to name a few. If the vehicle is set up with a weapon system, the Main Gun or Machine Gun settings allow the movement of the turret with the same controller. The Waypoint Follow button has a submenu that gives the user the control to initiate waypoint following. The navigation is set using the map screen to create a waypoint path for the vehicle to follow.

INTENTIONALLY LEFT BLANK.

Appendix C. Creating a New Mission: Load RIVET

1. **Open RIVET.** The Load Mission button leads the user to the next graphical user interface (GUI) shown in Fig. C-1 (see also the *RIVET Developer's Guide*,[1] p. 41–43). It allows the user to choose the mission, the player character (vehicle/Soldier), and whether it is friendly or enemy.

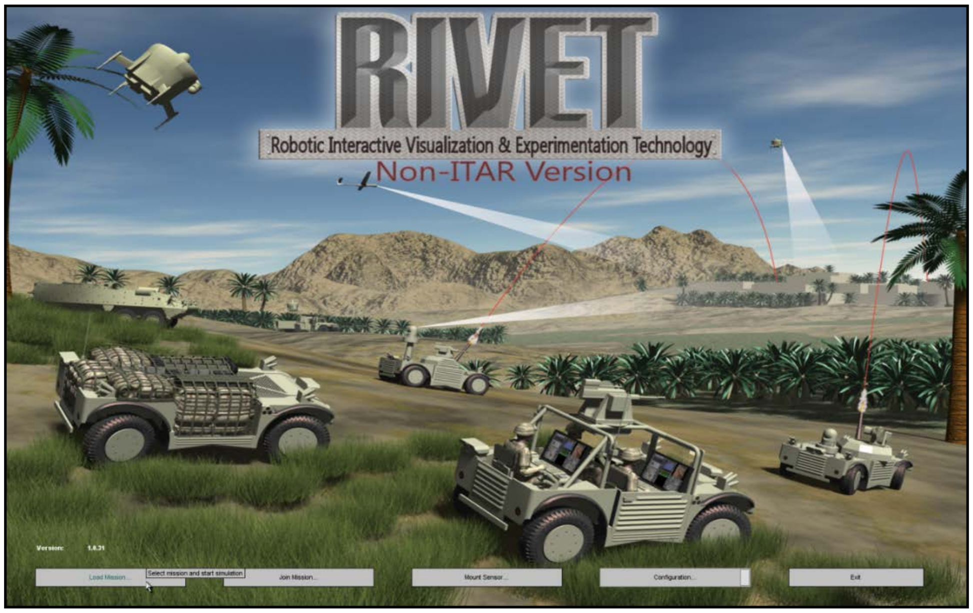
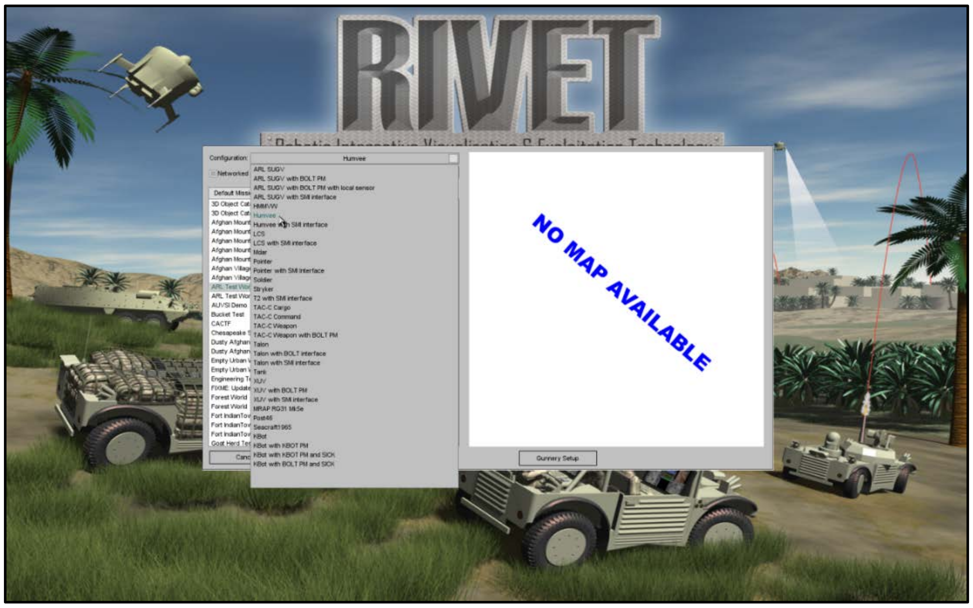


Fig. C-1 RIVET load mission menu

[1] General Dynamics Robotic Systems. RIVET developer's guide (included in software installer package in Docs folder). Westminster (MD): General Dynamics Robotics Systems; 2010. p. 41–43.

2. **Pick a player character.** When first building a new virtual environment (VE) for HRI research, it is recommended to select either a Soldier or Humvee (Fig. C-2). These two player characters are designed to function well with the keyboard and mouse user interface. However, when using RIVET with the BOLT sensor and CARVE application, choose only a vehicle with BOLT listed in the title. BOLT deactivates the keyboard and mouse associated with the Server computer.



Configuration: Humvee

Fig. C-2 Selecting a player character: Humvee

3. **Choose your mission.** When creating a new VE, it may be beneficial to begin by opening a Mission file that has already been created, as shown in Fig. C-3 (e.g., ARL Test World). The associated TorqueScript mission file (ARL Test World.mis) is located in the following directory: C:\RIVET\sim.base \data\missions.

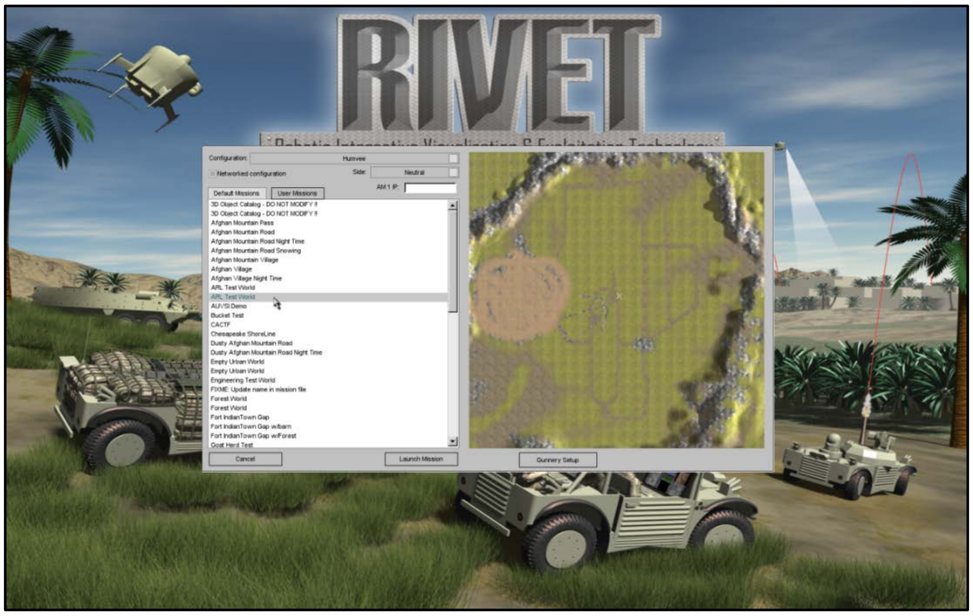

Configuration: Humvee
Networked configuration
Side: Neutral
AM 1 IP:

Fig. C-3 Selecting a mission

To successfully open and run a mission, this directory (C:\RIVET\sim.base\data\missions) should have the following files: Mission file (.mis) and Terrain file (.ter). To run BOLT and CARVE, the directory should also contain a Map (.png) and a location file (.cs).

4. **Launch Mission.** Commands for controlling the player character and switching camera views (first person, third person, or birds-eye view [Fig. C-4]) are available in the *RIVET Developer's Guide*[1] (p. 48–49).

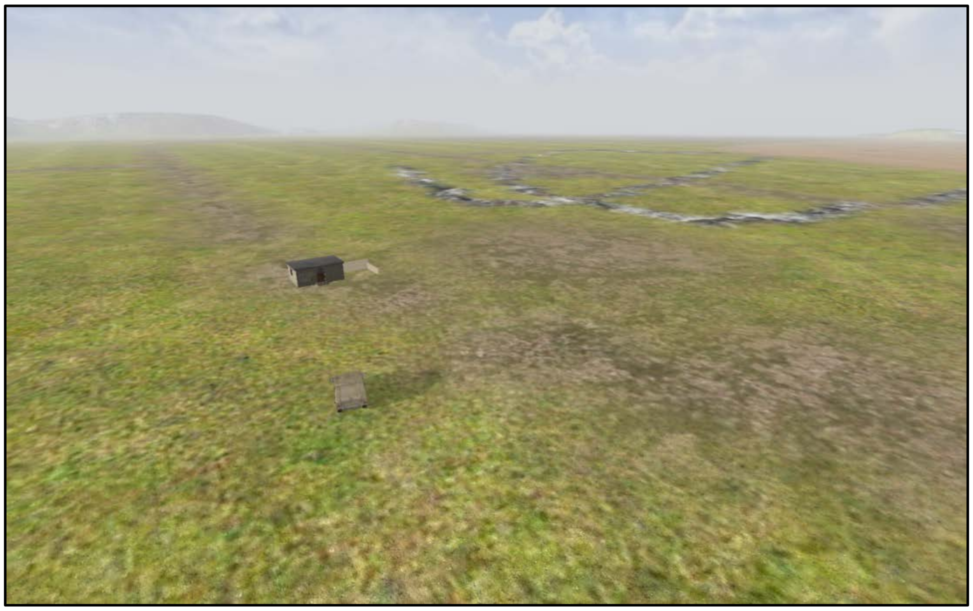


Fig. C-4 Changing camera views: example of a birds-eye view

5. **Save As a new Mission File.** Press F11 on the keyboard; click on File, and select Save Mission As (Fig. C-5).

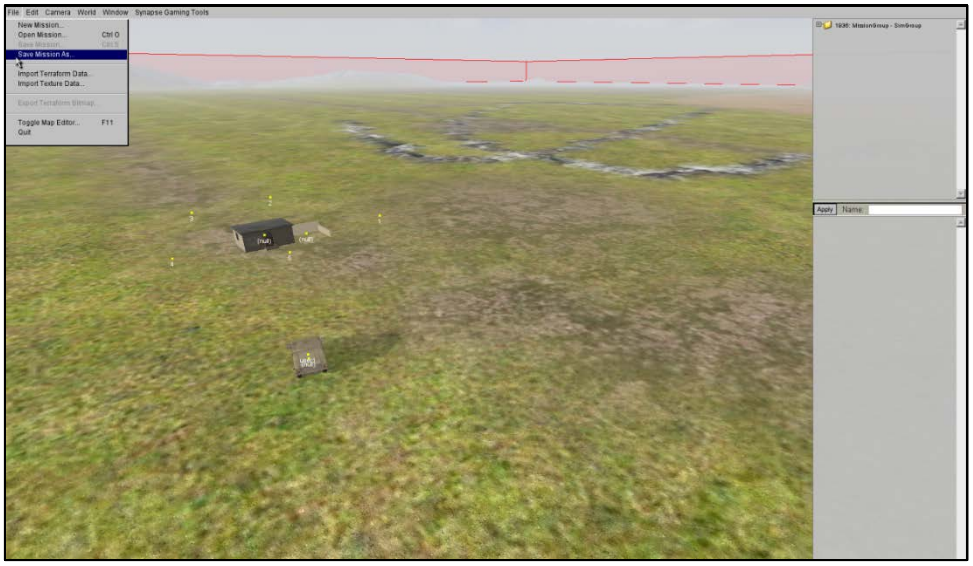
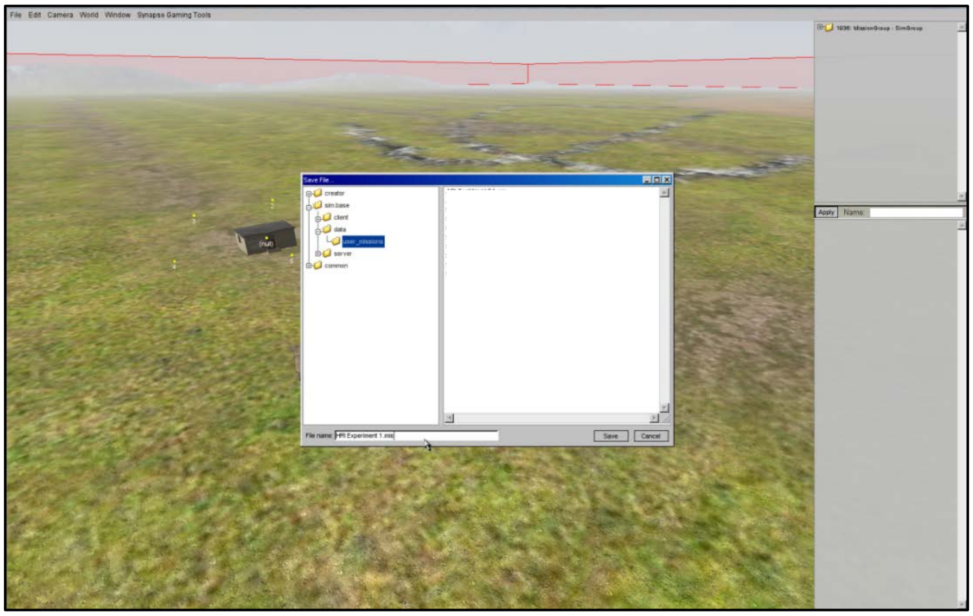


Fig. C-5 How to save a new mission file

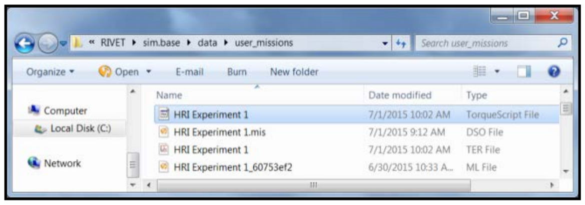
6. **Rename File.** This will save the mission in a new directory [C:\RIVET\sim.base\data\user_missions]. Make sure to rename the file (e.g., HRI Experiment 1.mis); otherwise it will overwrite a previous file (Fig. C-6).



File name: HRI Experiment 1.mis

Fig. C-6 Rename file

This process will create the Mission file (.mis) and a Terrain file (.ter). To check that the files were created, open the directory (C:\RIVET\sim.base \data\usermissions) as shown in (Fig. C-7).



Name	Date modified	Type
HRI Experiment 1	7/1/2015 10:02 AM	TorqueScript File
HRI Experiment 1.mis	7/1/2015 9:12 AM	DSO File
HRI Experiment 1	7/1/2015 10:02 AM	TER File
HRI Experiment 1_60753ef2	6/30/2015 10:33 A...	ML File

Fig. C-7 Save mission as new file name

7. **Save Mission.** Once you have created a new Mission file with a new name, it is important to save often. The mission can be saved by selecting file menu option (Press F11, select File, and select Save Mission). It is also possible to save the mission by using the keyboard command [Ctrl+s].

8. **Delete Current Objects (e.g., building and path).** To delete an object use the mouse, hold left mouse button down and drag the mouse to select the area desired. A yellow box will appear around the objects selected. To mark that the object has been selected, it will be outlined in red (Fig. C-8). To delete these objects, press the Delete key on the keyboard. Note there are no "undo" options.

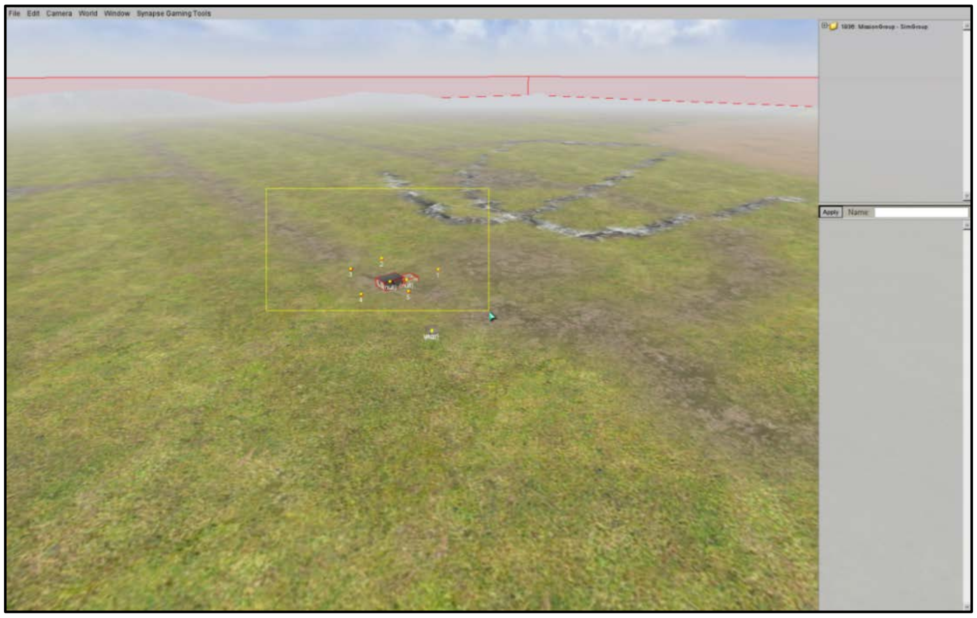
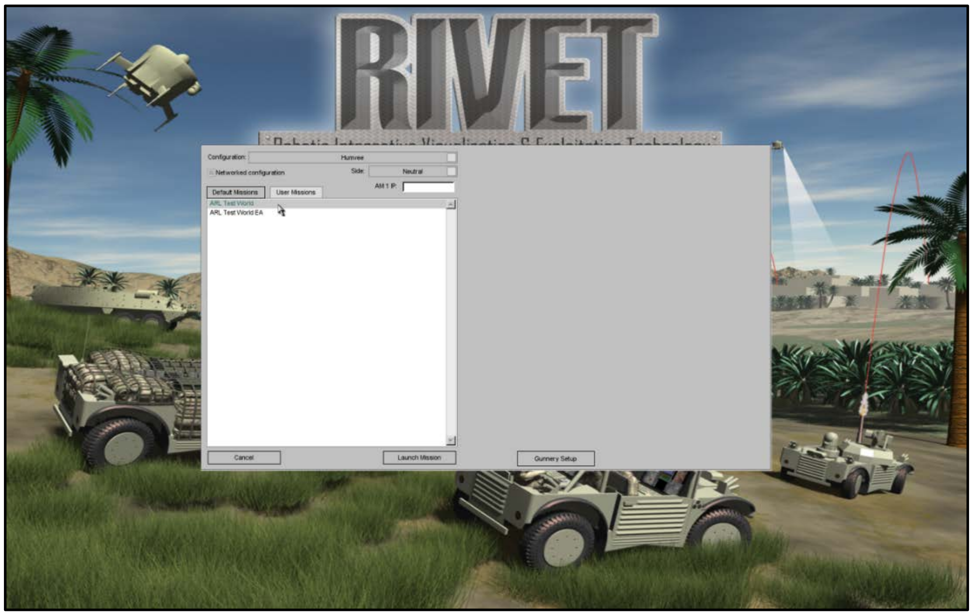


Fig. C-8 Selecting objects in the VE

9. **Closing RIVET.** There are different ways to close RIVET. The first is through the File menu (F11, File, Quit). The second is by using the keyboard command, Alt+F4. Remember to save before closing RIVET.

10. **Open Saved Mission.** To open the newly created User Mission file, open RIVET and click on Launch Mission (see step 1). Click on the User Mission tab and select the mission. As shown in Fig. C-9, the mission name will be the original file name (e.g., ARL Test World) even though the file was saved with a new name [HRI Experiment 1.mis]. This is still the newly created mission; however, there is an additional step to change the name of the file in the Main Menu GUI.



Configuration: Humvee
Networked configuration
Side: Neutral
AM 1 IP:

ARL Test World
ARL Test World EA

Fig. C-9 Open saved user mission

11. **Change User Mission File Name.** To update the file name in Main Menu GUI, close the RIVET Main Menu (press Cancel and Exit to close RIVET. The next step involves editing the TorqueScript. Open the Mission file (.mis) from the directory. Directions to access the .mis file in the Data Directory can be found in step 6. . Open the mission file by using Notepad, Notepad++, or Torsion. At the top of the .mis file is the TorqueScript for the "name" and "desc0" (see Fig. C-10 – Original Script). Both should be updated so that name matches the new User Mission name and desc0 marks the person(s) responsible for creating the new VE. These are marked in red text (see Fig. C-10 – Revised Script).

Original script	Revised script
new ScriptObject(MissionInfo) {	new ScriptObject(MissionInfo) {
name = "ARL Test World";	**name** = "RIVET TR Example: HRI Experiment 1";
desc0 = "General Dynamics Robotic Systems";	**desc0** = "Designed by Dr. K.E. Schaefer, ARL";
aiPlayerFile = "testAIplayer.cs";	aiPlayerFile = "testAIplayer.cs";
descLines = "1";	descLines = "1";
map = "ARL_world";	map = "ARL_world";
};	};

Fig. C-10 Example TorqueScript to update the user mission name for the main menu GUI

12. Now when RIVET is opened and Launch Mission selected, the User Missions tab will list the updated name of the correct user mission (Fig. C-11). While this will not inhibit use of the system, updating the User Mission file name in the Main Menu GUI is encouraged, as it helps to accurately and effectively differentiate between multiple User Mission files.

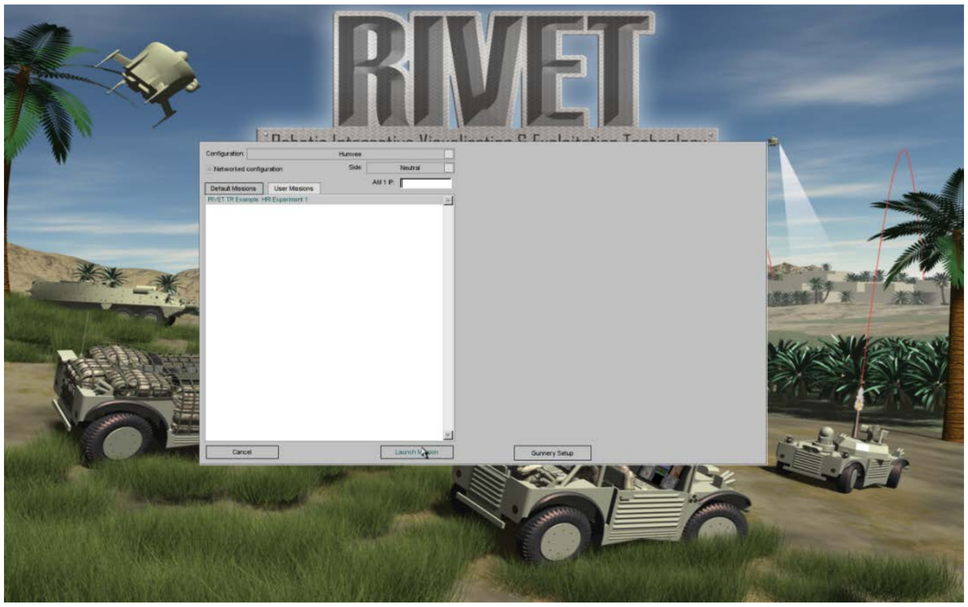
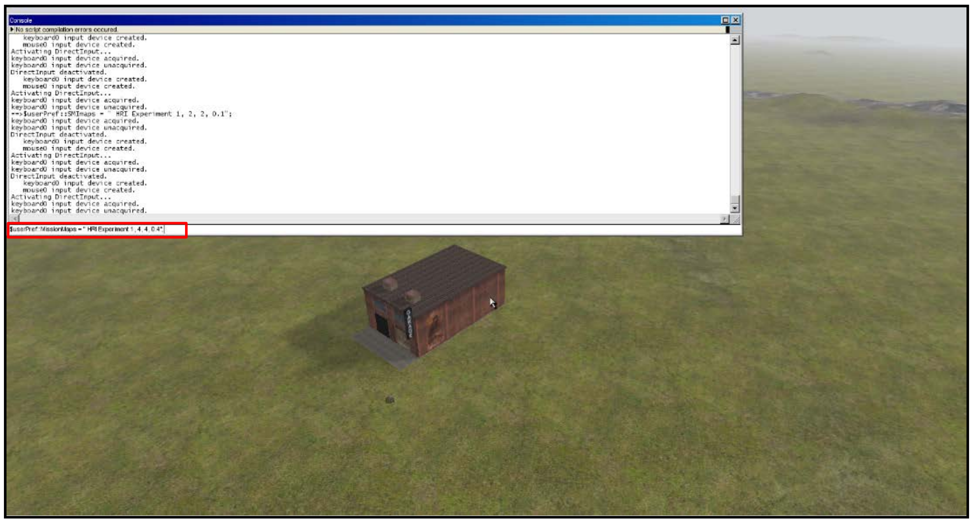


Fig. C-11 Main menu GUI: updated user mission file name

13. **Map.** Another way to help differentiate between the user missions is by adding a map to the User Mission login screen. Map creation is also essential when using the CARVE application. To create a Map, first load RIVET and launch the User Mission. Next, press Alt+c on the keyboard to switch camera view. Use W, A, S, D keyboard commands at the same time as moving the mouse to move upward into the air.

14. **Creating the Map.** To initiate the process for creating a map, first follow the steps to name the map the same name as the User Mission. Press the tilde (~) key on the keyboard to open the Console and type $userPref::MissionMaps = "HRI Experiment 1, 4, 4, 0.4"; as shown in Fig. C-12. Next, press Enter on the keyboard and ~ to close the Console. To create the Map, press Alt+m on the keyboard. The Map (.png) and associated location file (.cs) are automatically place into the RIVET directory C:\RIVET (Fig. C-13).



```
▶No script compilation errors occured.
    keyboard0 input device created.
    mouse0 input device created.
Activating DirectInput...
keyboard0 input device acquired.
keyboard0 input device unacquired.
DirectInput deactivated.
    keyboard0 input device created.
    mouse0 input device created.
Activating DirectInput...
keyboard0 input device acquired.
keyboard0 input device unacquired.
==>$userPref::SMImaps = " HRI Experiment 1, 2, 2, 0.1";
keyboard0 input device acquired.
keyboard0 input device unacquired.
DirectInput deactivated.
    keyboard0 input device created.
    mouse0 input device created.
Activating DirectInput...
keyboard0 input device acquired.
keyboard0 input device unacquired.
DirectInput deactivated.
    keyboard0 input device created.
    mouse0 input device created.
Activating DirectInput...
keyboard0 input device acquired.
keyboard0 input device unacquired.
$userPref::MissionMaps = " HRI Experiment 1, 4, 4, 0.4";
```

Fig. C-12   Create map screen

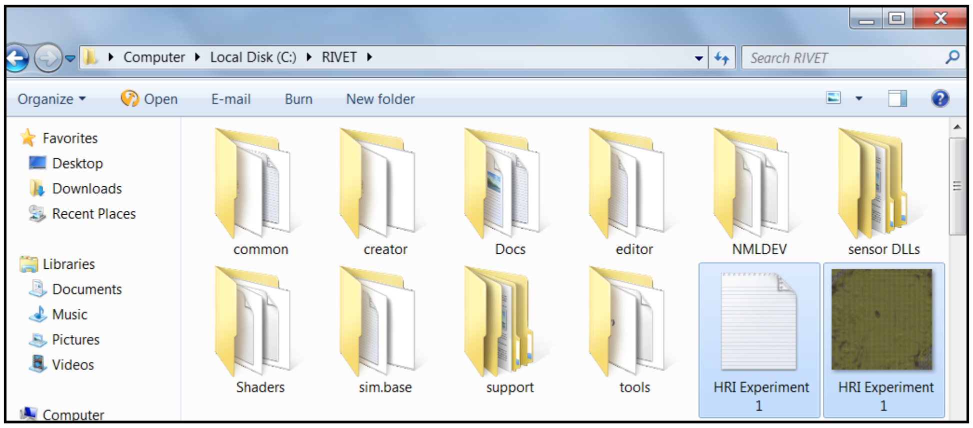
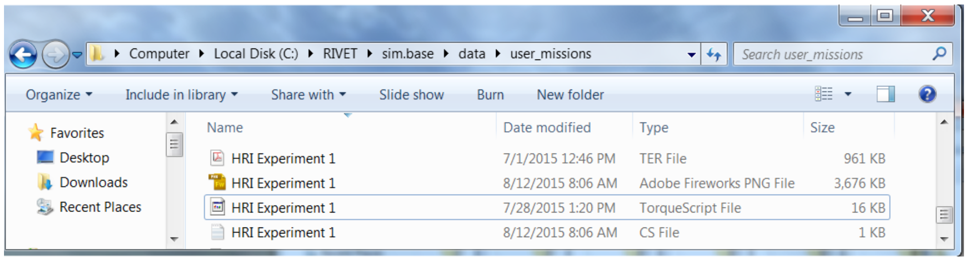


Fig. C-13 RIVET directory location for map (.png) and location (.cs) files

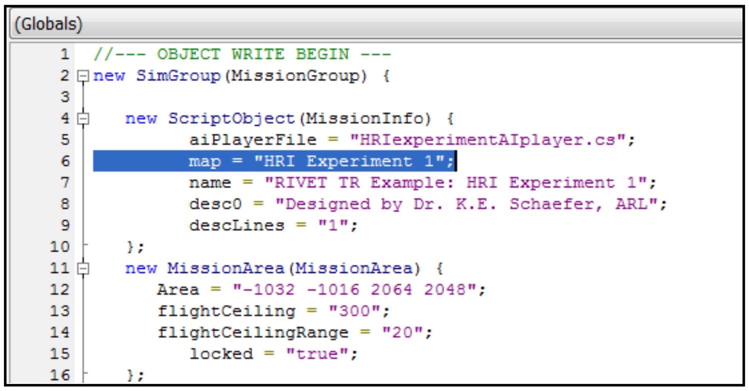
15. **Setting up a Map in RIVET.** For RIVET to be able to successfully read the Map, first, move the map and location files (.png and .cs) to the User Mission directory as shown in Fig. C-14 (C:\RIVET\sim.base\data\usermissions).


Computer ▸ Local Disk (C:) ▸ RIVET ▸ sim.base ▸ data ▸ user_missions

Name	Date modified	Type	Size
HRI Experiment 1	7/1/2015 12:46 PM	TER File	961 KB
HRI Experiment 1	8/12/2015 8:06 AM	Adobe Fireworks PNG File	3,676 KB
HRI Experiment 1	7/28/2015 1:20 PM	TorqueScript File	16 KB
HRI Experiment 1	8/12/2015 8:06 AM	CS File	1 KB

Fig. C-14 User mission directory

Next, open up the Mission file (C:\RIVET\sim.base\data\usermissions\HRI Experiment 1.mis) and add or revise the TorqueScript (see Fig. C-15, line 6). This will now call the Map. When RIVET is opened, select the User Mission, and a Map will now be present (Fig. C-16).


(Globals)

```
 1  //--- OBJECT WRITE BEGIN ---
 2  new SimGroup(MissionGroup) {
 3
 4     new ScriptObject(MissionInfo) {
 5           aiPlayerFile = "HRIexperimentAIplayer.cs";
 6           map = "HRI Experiment 1";
 7           name = "RIVET TR Example: HRI Experiment 1";
 8           desc0 = "Designed by Dr. K.E. Schaefer, ARL";
 9           descLines = "1";
10     };
11     new MissionArea(MissionArea) {
12        Area = "-1032 -1016 2064 2048";
13        flightCeiling = "300";
14        flightCeilingRange = "20";
15           locked = "true";
16     };
```

Fig. C-15 TorqueScript: adding the map

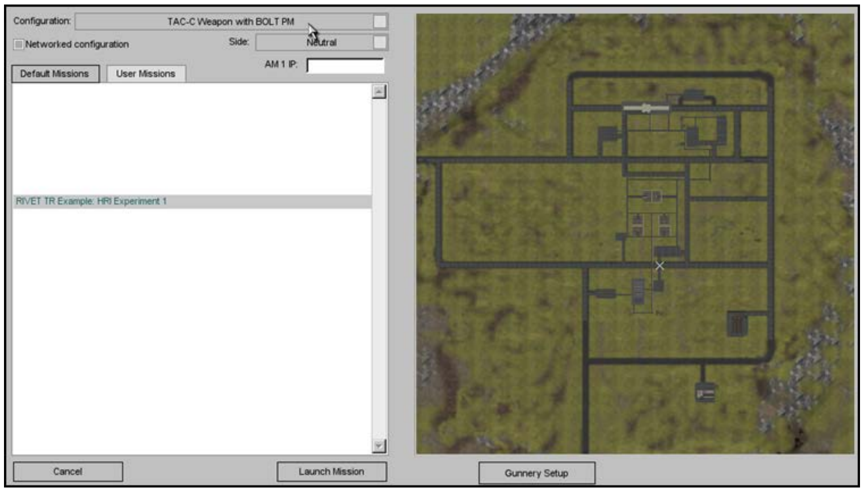
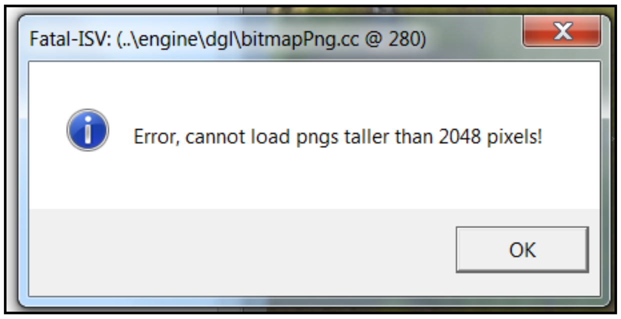

Configuration: TAC-C Weapon with BOLT PM
Networked configuration
Side: Neutral
AM 1 IP:
RIVET TR Example: HRI Experiment 1

Fig. C-16 Main Menu GUI: user mission file with map

16. **Map Size.** To check the map size, right click on the map to look at the properties of the file. Ideally, the map should be 1600 × 1600 pixels and under 2 MB in size. If this is not the case, use an outside program (e.g., Adobe Fireworks) to reduce the size of the map (.png). If this step is missed, RIVET will provide an error message as shown in Fig. C-17.

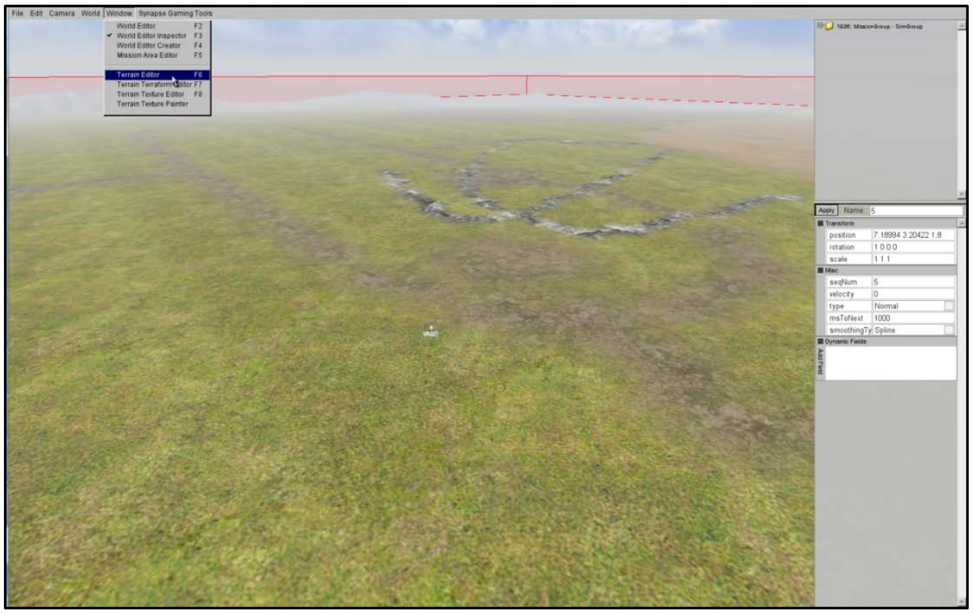

Fatal-ISV: (..\engine\dgl\bitmapPng.cc @ 280)

Error, cannot load pngs taller than 2048 pixels!

Fig. C-17 Error message

Appendix D. Creating a New Mission: Editing the Terrain

1. **Terrain Editor Tool.** Use the Terrain Editor tool to adapt the height and shape of the terrain (Fig. D-1). It can be accessed by pressing F11 on the keyboard and selecting Window and Terrain Editor from the drop-down menu.



Name: 5
position 7.18994 3.20422 1.8
rotation 1 0 0 0
scale 1 1 1
seqNum 5
velocity 0
type Normal
msToNext 1000
smoothingTy Spline

Fig. D-1 Terrain Editor tool

2. **Action Menu.** Select the action (e.g., add dirt, adjust height) to be revised (Fig. D-2).

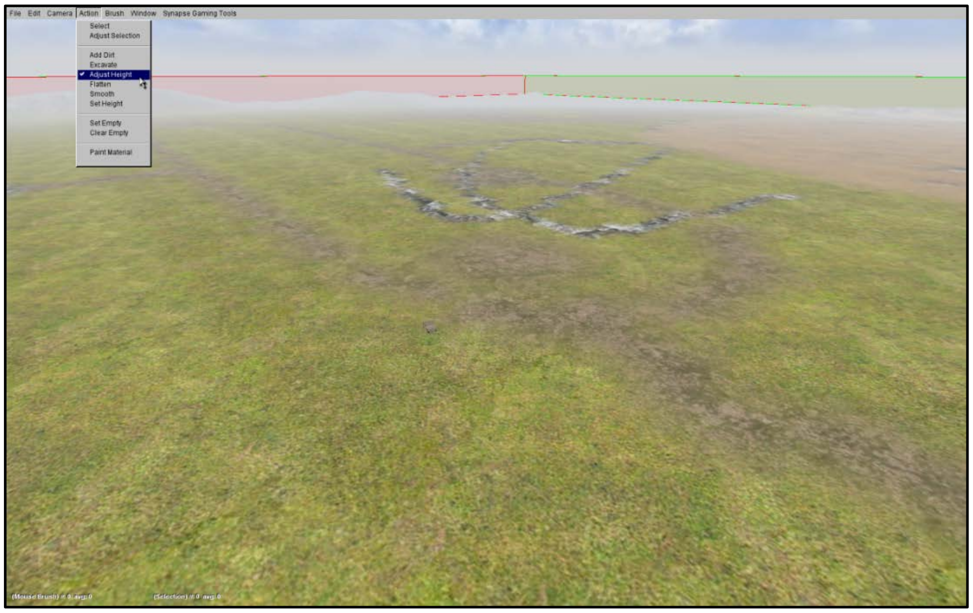


Fig. D-2 Terrain Editor: Action submenu

3. **Brush size.** Choose the size of the brush (Fig. D-3).

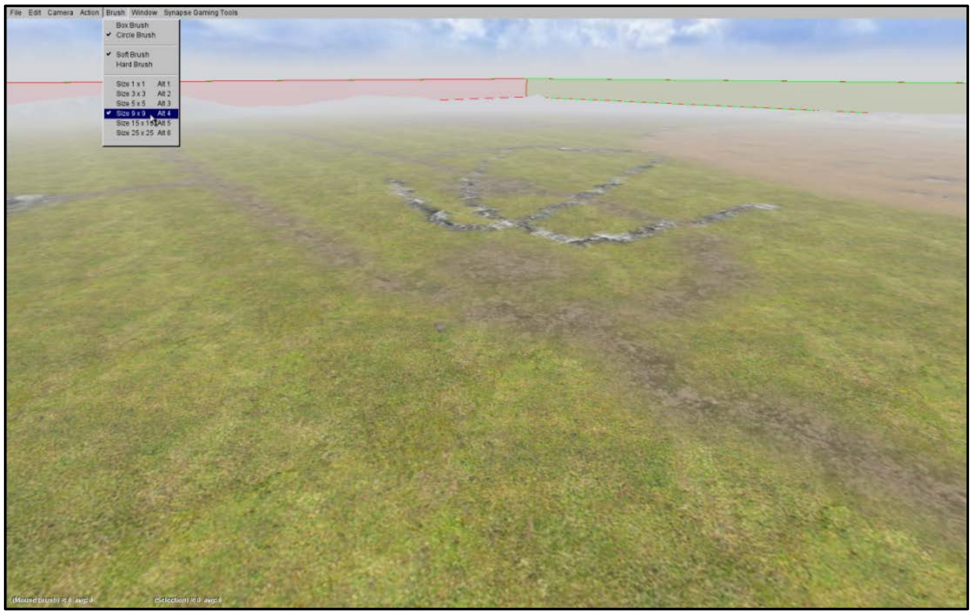


Fig. D-3 Terrain Editor: Brush submenu

4. To edit the shape of the terrain, use the mouse. Click and hold the left mouse button while dragging left, right, up, or down (Fig. D-4). This will raise or lower the ground terrain.

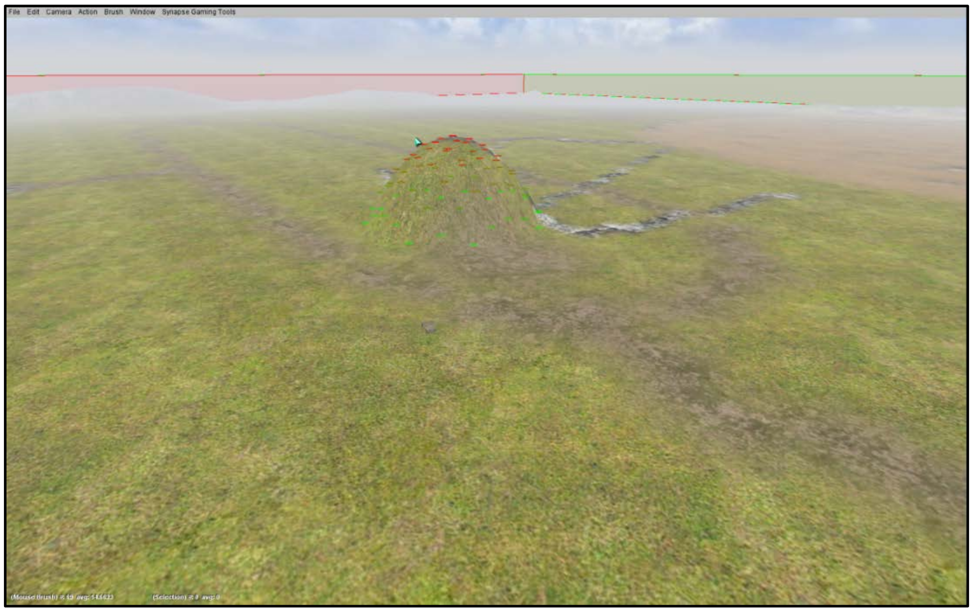


Fig. D-4 Edit the shape of the terrain

5. **Terrain Painter tool.** This tool bar is used to change the underlying texture of the terrain. Click on one of the terrains and then paint the environment (Fig. D-5).

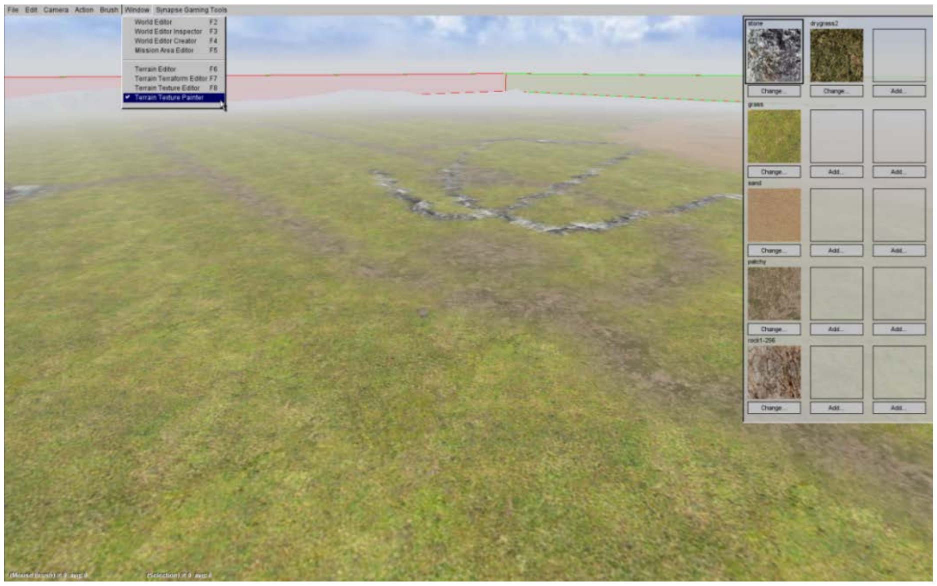


Fig. D-5 Terrain Painter tool

It is possible to add additional textures by clicking on Add (Fig. D-6, right side menu) and searching through the menu options for different image files (.jpg) (Fig. D-7). These image files can also be found in the terrains directory, C:\RIVET\sim.base\data\terrains\terrains\snow2.jpg. This allows for greater flexibility in the design of the terrain that can be updated at any time during creation of the virtual environment.

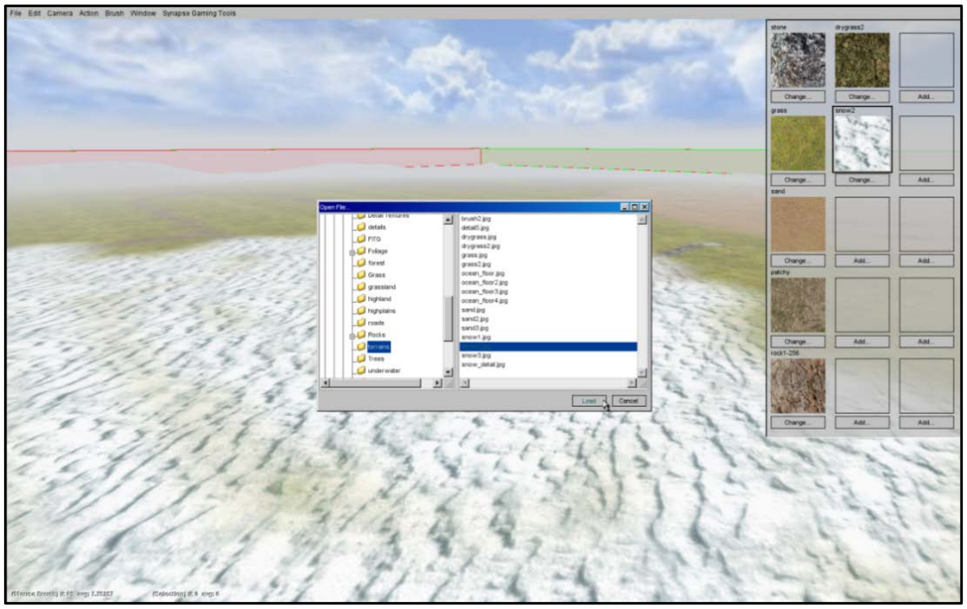


Fig. D-6 Texture submenu

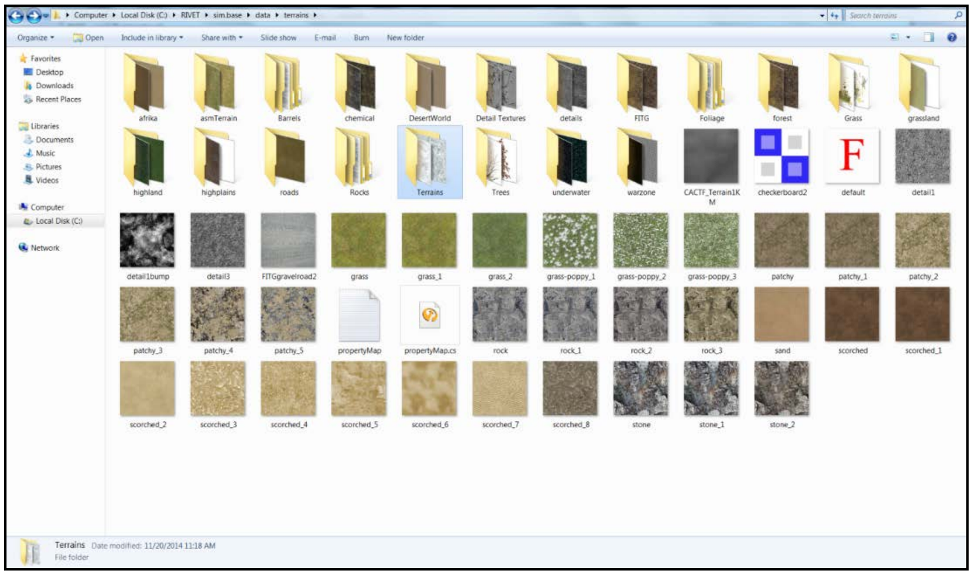


Fig. D-7 Available terrain textures

6. **Save often!**

INTENTIONALLY LEFT BLANK.

Appendix E. Add Static Objects

1. **Change Camera View.** Currently, the virtual environment is very empty (Fig. E-1). Before adding objects, it is important to change the camera's viewing angle.

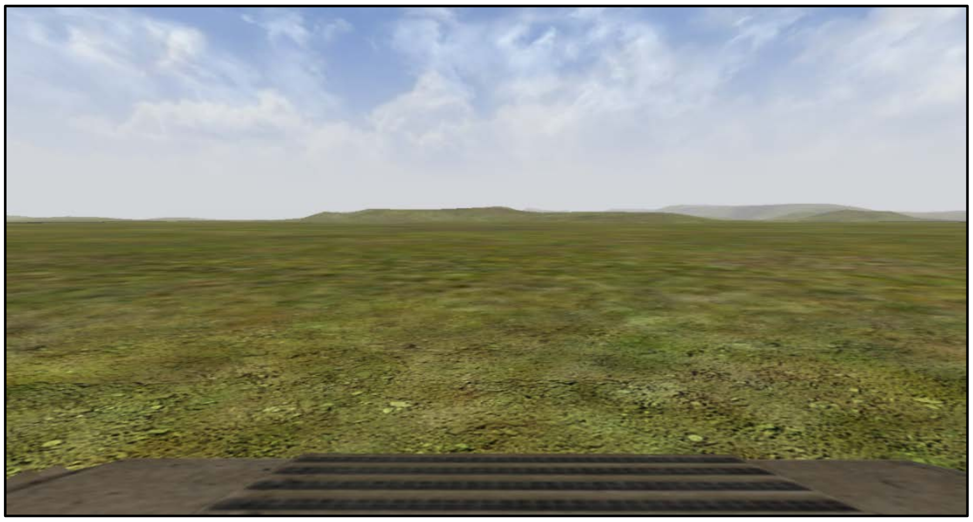


Fig. E-1 Original view

To toggle your camera to Fly mode (also known as birds-eye mode), press Alt+c on the keyboard. This same action can also be achieved through the menu options (F11) by selecting the Camera menu and the Toggle Camera option. After the camera is placed in Fly mode, pull the mouse toward you and hold down the "s" key. It will appear that you are "flying" higher into the air while looking down at the ground. It should be possible to see the Humvee on the ground below (Fig. E-2). This is the best view to have when adding static objects; otherwise, they could end up being placed underground.

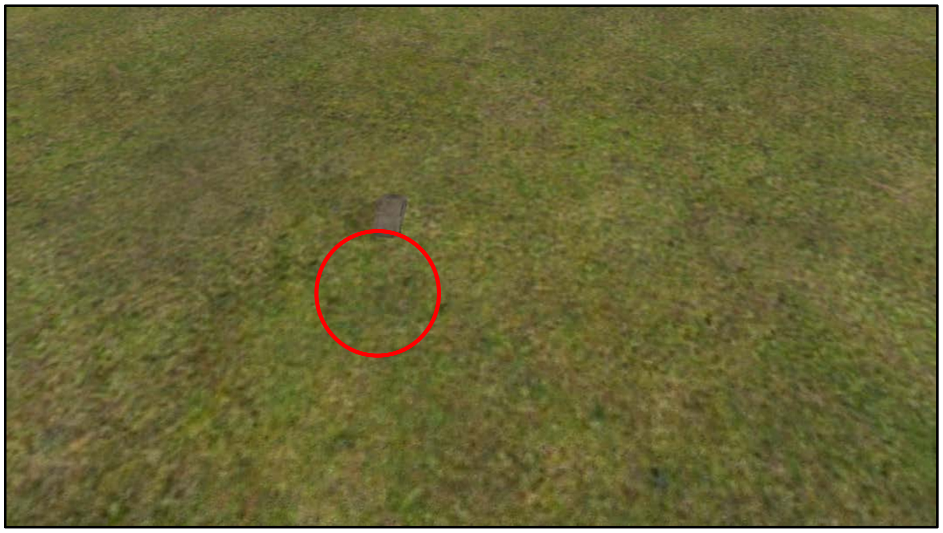


Fig. E-2 Fly mode

2. **Add objects.** Use the World Editor Creator to add objects to the VE (Fig. E-3). To open the World Editor Creator, press F11, select Window and World Editor Creator. It may also be possible to open this menu by pressing F4 on the keyboard.

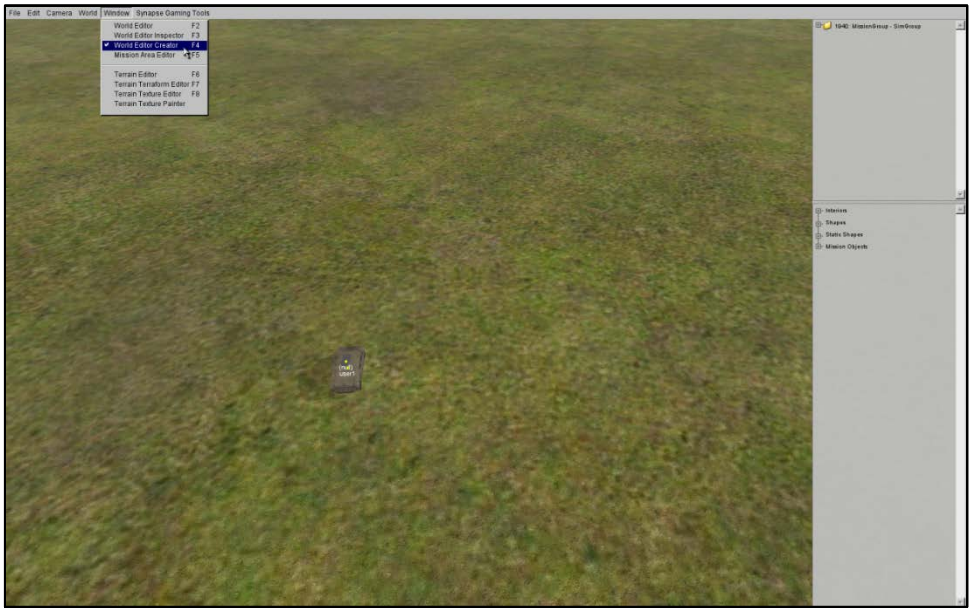


Fig. E-3 World Editor Creator tool to add objects

3. **Static Objects.** RIVET already has a number of static objects available for use. Most of these are available under the Interiors or Static Shapes directory (Fig. E-4).

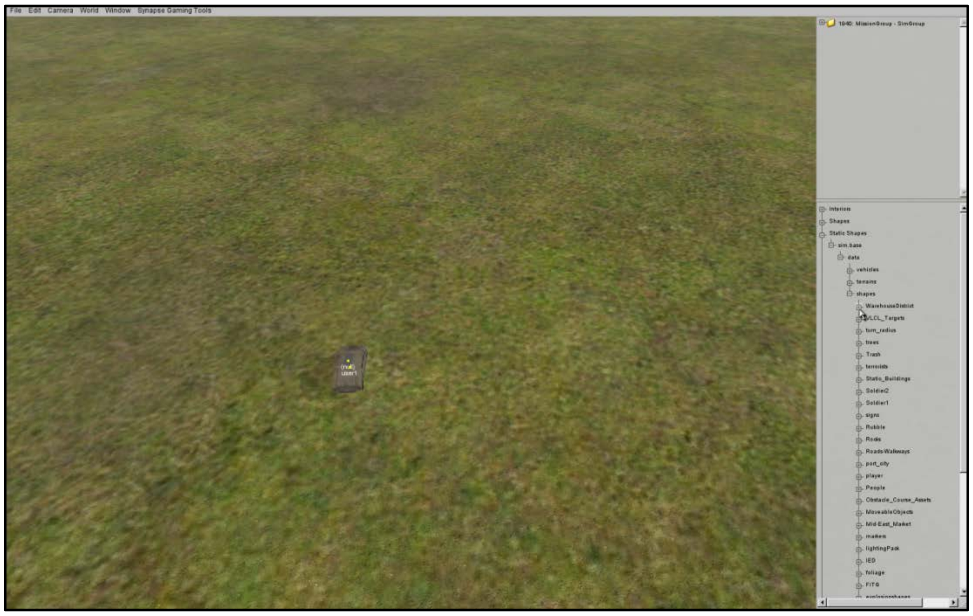


Fig. E-4 RIVET static object directory

4. **Finding Static Objects.** It may be difficult to know what static objects are currently available. RIVET has a Mission File [3D Object Catalog - DO NOT MODIFY!!!] that provides a single location that includes a number of (but not all) of the static objects available (e.g., buildings, roadways, warehouse objects, and street signs). Fig. E-5 depicts the Main Menu GUI options for opening the 3D Object Catalog file. Figures E-6 through E-8 provide examples of some the static objects available.

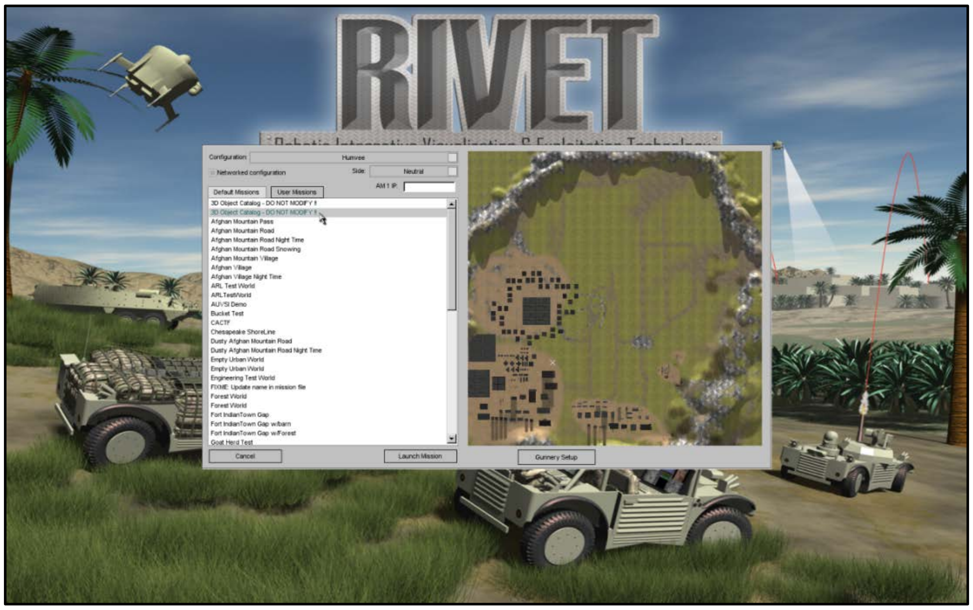


Fig. E-5 3D Object Catalog VE located in the Main Menu GUI

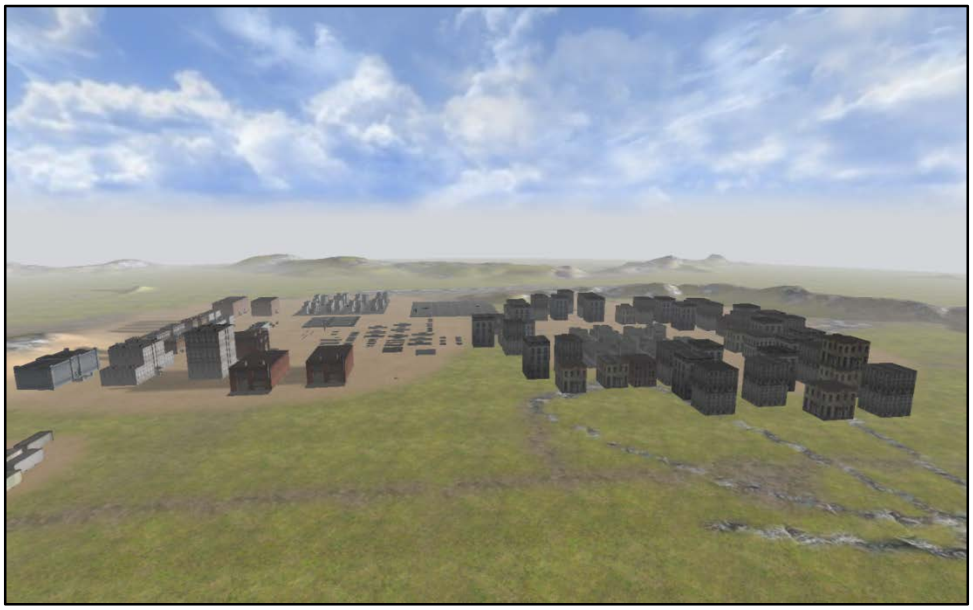


Fig. E-6 Options for static buildings

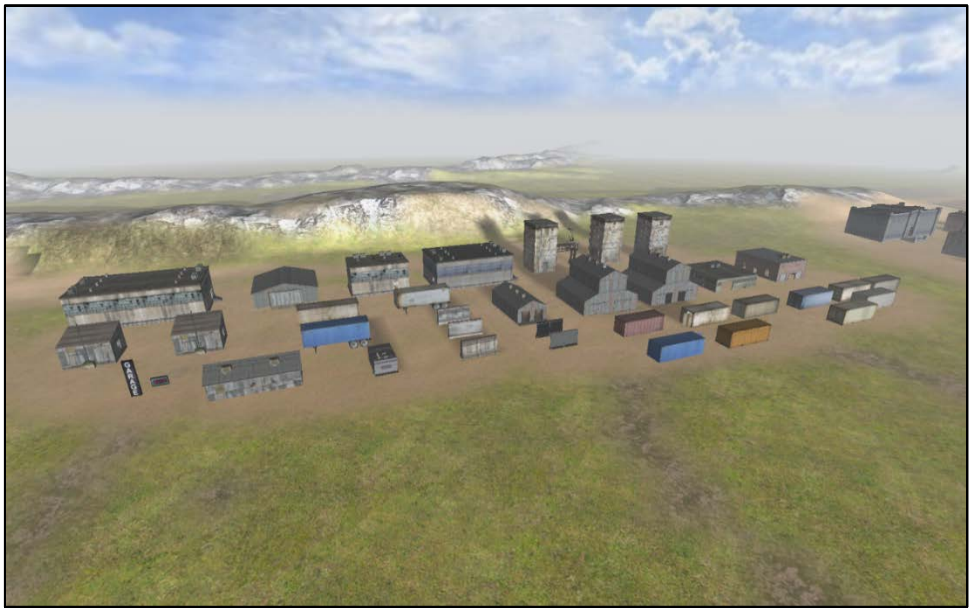


Fig. E-7 Options for static warehouses

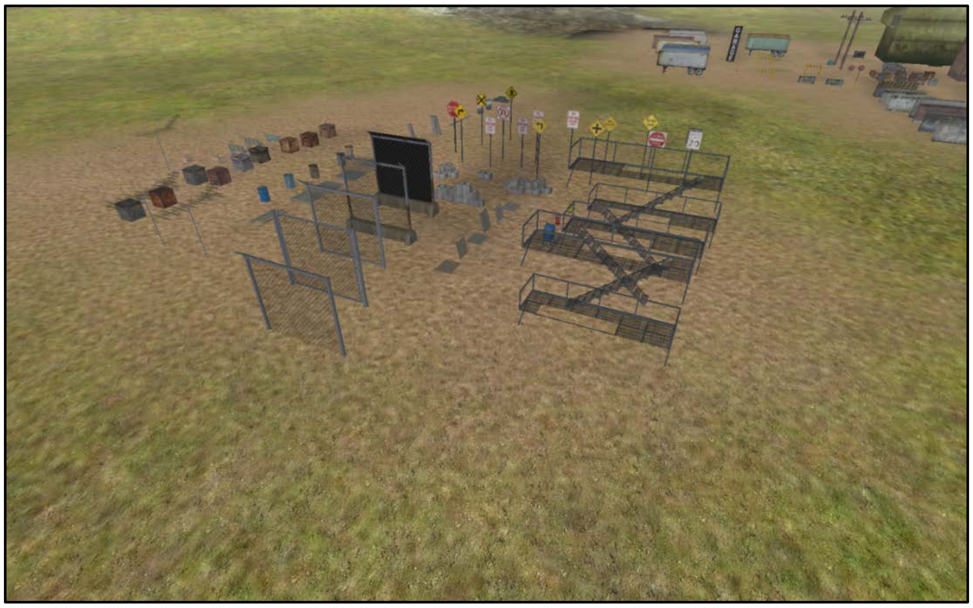
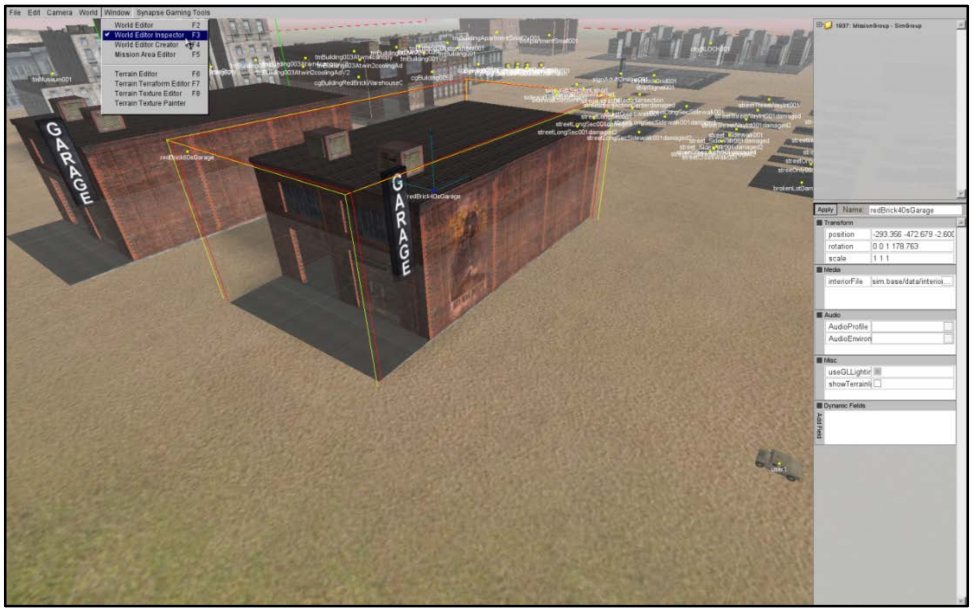


Fig. E-8 Options for static signs and other obstacles

5. **Locating an object's file location.** To find the location of an object in the tree structure for later use in the VE, press F11, click on desired object, and open the World Editor Inspector (Fig. E-9). The World Editor Inspector can be opened by either pressing the F3 key on the keyboard, or pressing F11, selecting Windows and World Editor Inspector from the drop-down menu.



Name:	redBrick40sGarage
position	-293.356 -472.679 -2.600
rotation	0 0 1 178.763
scale	1 1 1
interiorFile	sim.base/data/interio...
AudioProfile	
AudioEnviron	
useGLLightir	▨
showTerrainI	☐

Fig. E-9 World Editor Inspector menu

Approved for public release; distribution is unlimited.

6. **World Editor Inspector.** Click on open file dialog box (mouse cursor, outlined with a red box) and then search for the name of the object (e.g., redBrick40sGarage) in the list (Fig. E-10). Make a note that this building is located in the Interiors\DowntownDistrict directory.

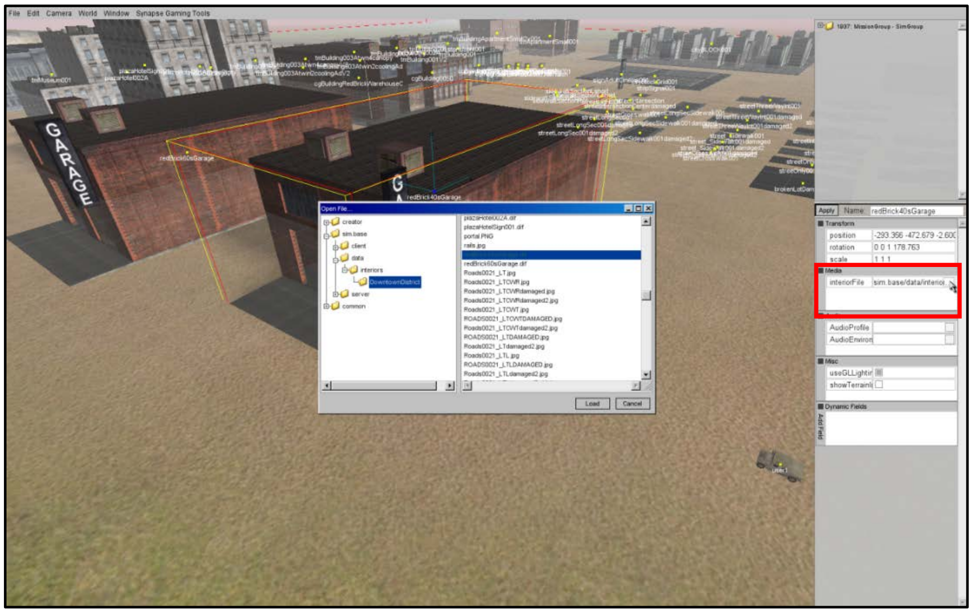


Fig. E-10 Locating an object file in the World Editor Inspector

7. Open the selected mission [RIVET TR Example: HRI Experiment 1], and repeat step 1 for adding static objects. The screen should appear as shown in Fig. E-11.

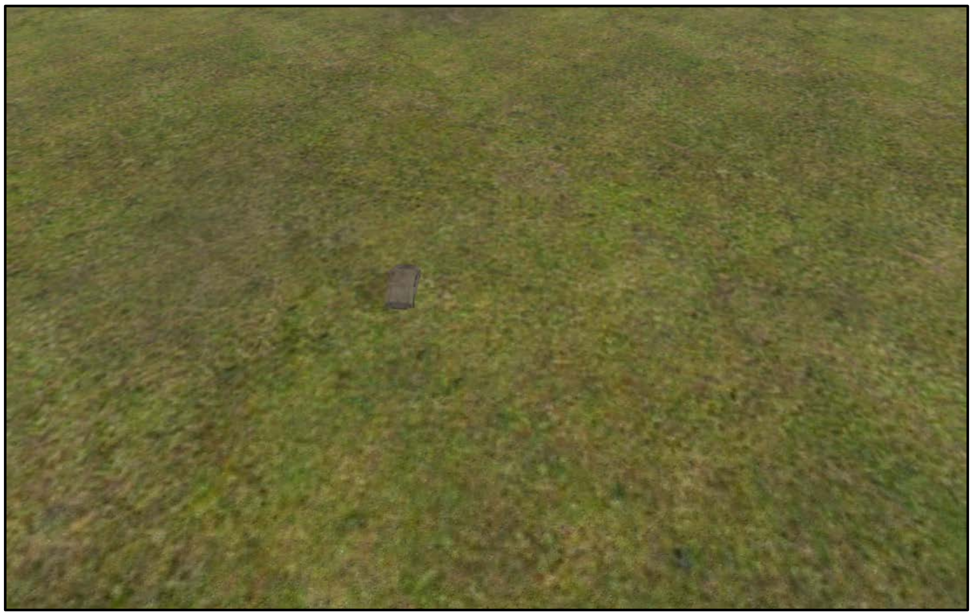


Fig. E-11 User VE from fly mode

Add a Static Object. Open the World Editor (F11) and then open the World Editor Creator (see steps 2 and 3). Open the Interiors menu (Fig. E-12).

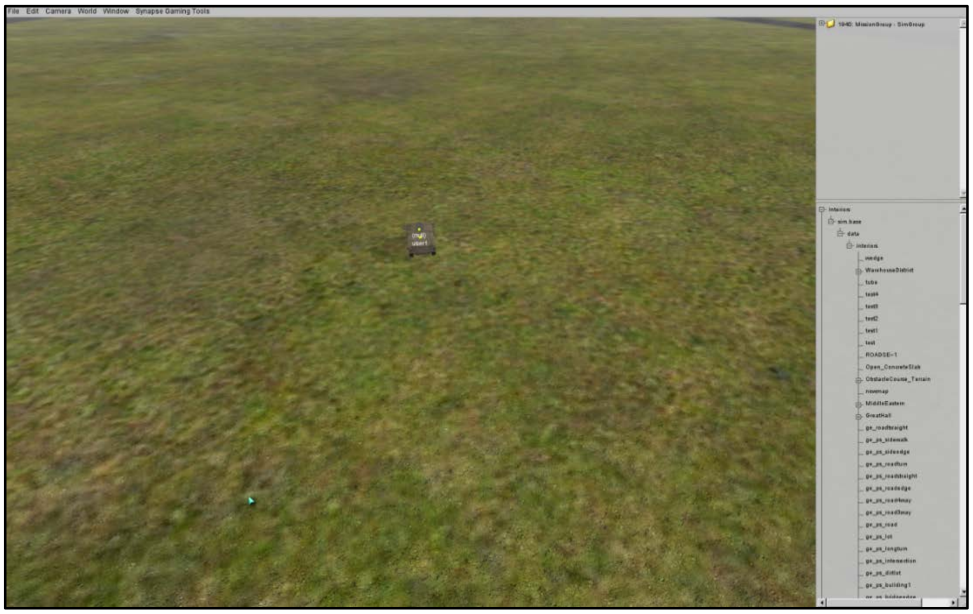


Fig. E-12 Adding a static object through the Interiors menu

8. **Finding the Garage.** Scroll down to find DowntownDistrict and click on redBrick40sGarage. Once the garage is selected, it will appear in the VE. However, as shown in Fig. E-13, the building is located on top of the Humvee and partially underground.

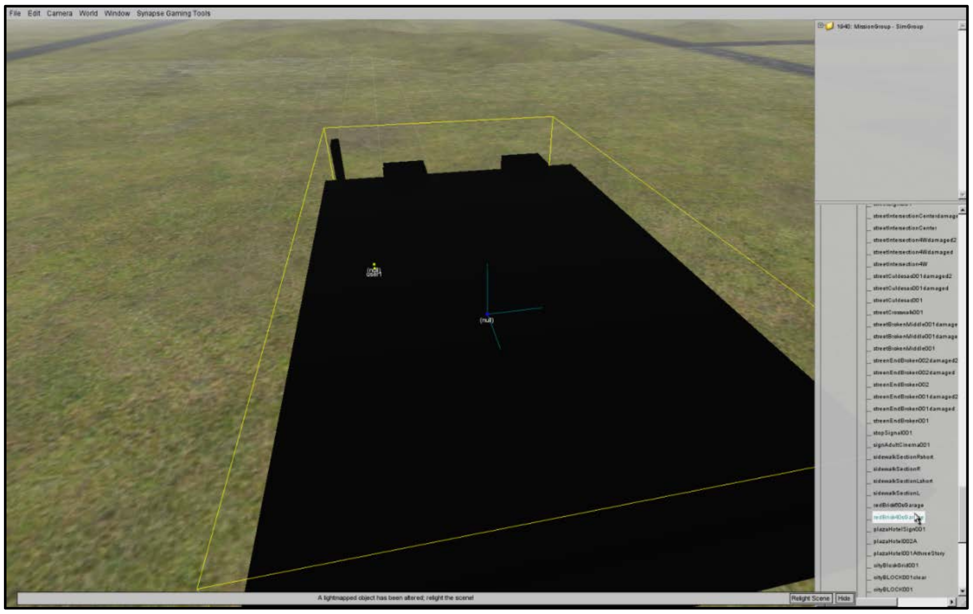
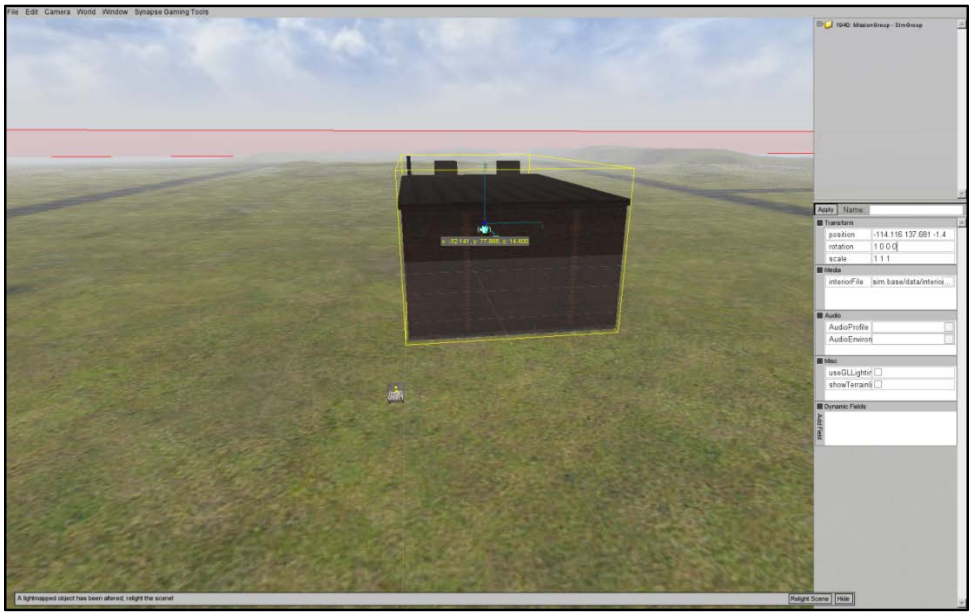

File Edit Camera World Window Synapse Gaming Tools
A lightmapped object has been altered; relight the scene!
Relight Scene
Hide

Fig. E-13 Initial building placement

Approved for public release; distribution is unlimited.

9. **Moving an object.** To move an object, click (on the X, Y, or Z-axis) and drag the building to a desired location (Fig. E-14).

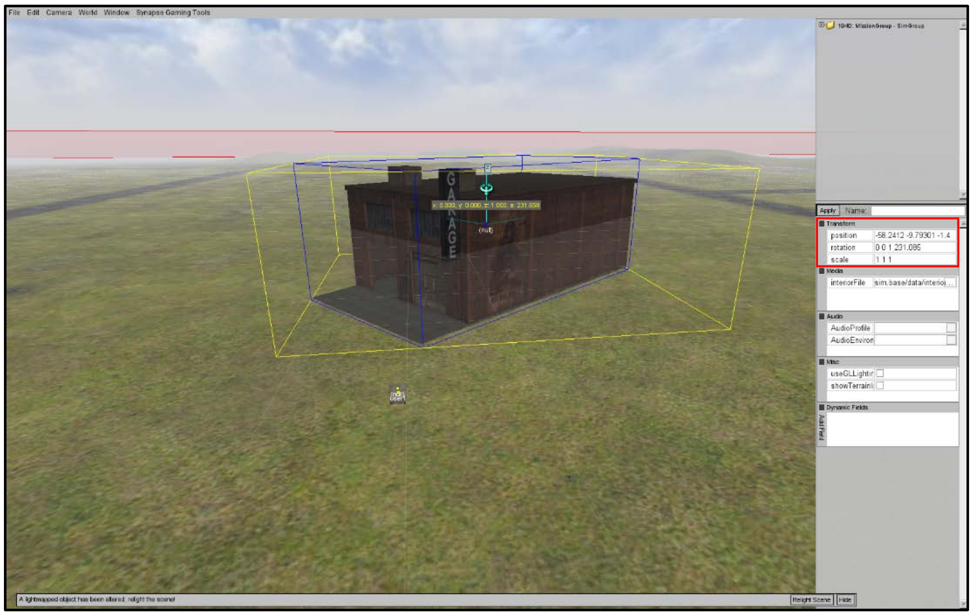

File Edit Camera World Window Synapse Gaming Tools
1940: MissionGroup - SimGroup
Apply Name:
Transform
Media
Audio
Misc
Dynamic Fields
Add Field
Relight Scene
Hide

position -114.116 137.681 -1.4
rotation 1 0 0 0
scale 1 1 1
interiorFile sim.base/data/interioi...
AudioProfile
AudioEnviron
useGLLightir ☐
showTerraini ☐

A lightmapped object has been altered; relight the scene!

Fig. E-14 Moving an object within the VE

10. **Rotating an object.** To rotate an object, press Alt on the keyboard and select the Z-axis with the mouse (left mouse button). While holding both buttons, move the mouse. The object will rotate. The position, rotation, and size of the object can also be changed using the World Editor Inspector (F3) menu (Fig. E-15).



position -58.2412 -9.79301 -1.4
rotation 0 0 1 231.085
scale 1 1 1
interiorFile sim.base/data/interioi...
AudioProfile
AudioEnviron
useGLLightir ☐
showTerraini ☐

A lightmapped object has been altered; relight the scene!

Fig. E-15 Rotating an object

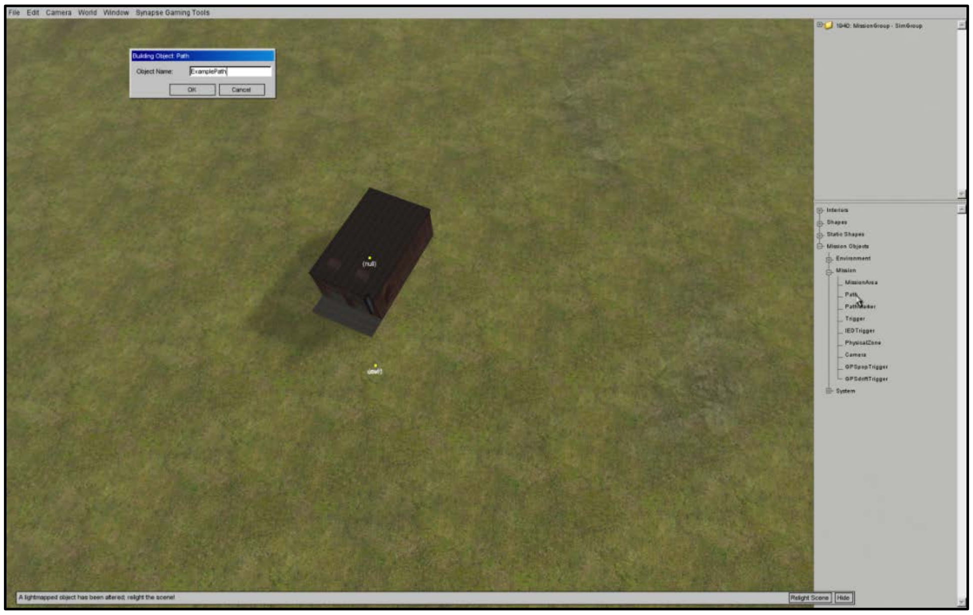
11. Keep adding objects to the VE until it resembles the experimental environment. These objects are automatically added to the Mission file (.mis). An example of the TorqueScript that is added into the .mis file for a static object (redBrick40sGarage.dif) is provided below.

12. **Save often!**

```
new InteriorInstance() {
    position = "-58.2412 -9.79301 -1.4";
    rotation = "0 0 1 231.085";
    scale = "1 1 1";
    interiorFile = "~/data/interiors/DowntownDistrict/redBrick40sGarage.dif";
    useGLLighting = "0";
    showTerrainInside = "0";
};
```

Appendix F. Creating a Path

1. **Adding a path.** To add a path that another vehicle or person may follow, open the World Editor Creator (F4). Click on Path to name the path. Assign a unique name to help differentiate between the paths. In Fig. F-1, the path name is ExamplePath.

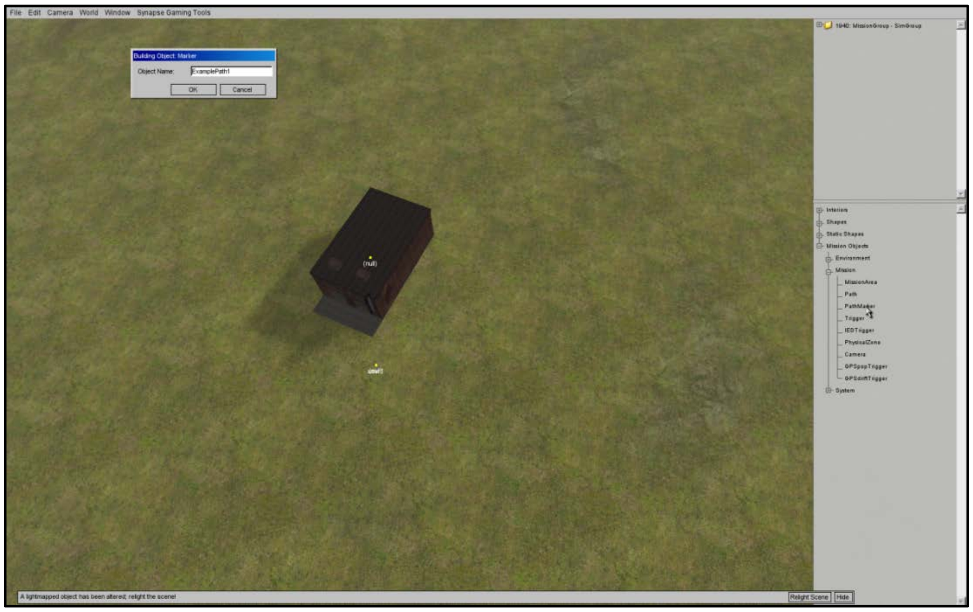


Object Name: ExamplePath

A lightmapped object has been altered; relight the scene!

Fig. F-1 Naming a new path

2. **Adding a PathMarker.** A PathMarker is used to label the points on the path (Fig. F-2). This is the process to add all the path markers needed to complete the desired path. Make sure every PathMarker has a unique name. Label names should not have any spaces (use underscore if needed). Each PathMarker can be moved in the VE in the same way static objects were moved. The Y-axis should face the direction in which the vehicle is to move for each PathMarker.



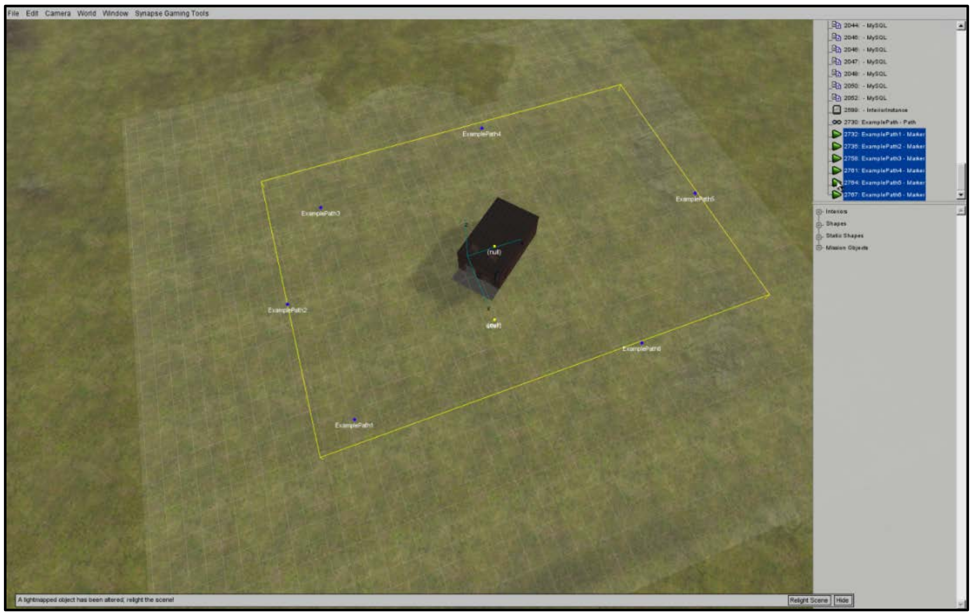
Building Object: Marker

Object Name: ExamplePath1

A lightmapped object has been altered; relight the scene!

Fig. F-2   Adding a PathMarker to the new path

3. **Linking the PathMarkers.** Now that all the PathMarkers are placed and rotated, select all the PathMarkers from the tree in the upper-right corner (Folder name +MissionGroup – SimGroup) (Fig. F-3). They should be located at the bottom of the tree structure since they were the most recent thing added to the VE. To select multiple PathMarkers at the same time, hold down the Ctrl key on the keyboard and click on each marker individually.

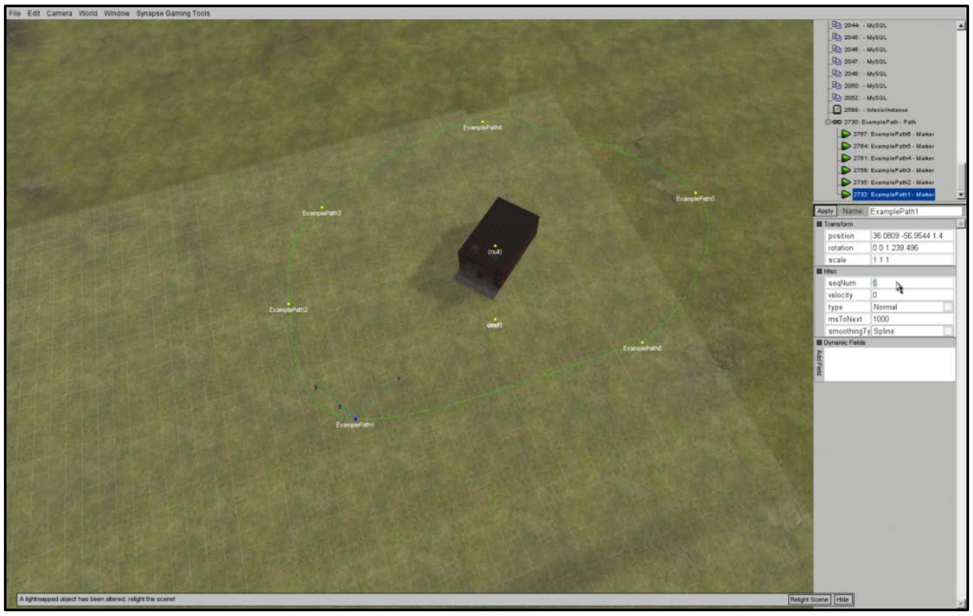


2044: - MySQL
2045: - MySQL
2046: - MySQL
2047: - MySQL
2048: - MySQL
2050: - MySQL
2052: - MySQL
2590: - InteriorInstance
2730: ExamplePath - Path
2732: ExamplePath1 - Marker
2735: ExamplePath2 - Marker
2758: ExamplePath3 - Marker
2761: ExamplePath4 - Marker
2764: ExamplePath5 - Marker
2767: ExamplePath6 - Marker

A lightmapped object has been altered; relight the scene!

Fig. F-3 Selecting all PathMarkers on a path

4. Move the PathMarkers under the path name (next to the infinity sign, see Fig. F-4). In the VE, all markers will be connected by a dotted green line.



2044: - MySQL
2045: - MySQL
2046: - MySQL
2047: - MySQL
2048: - MySQL
2050: - MySQL
2052: - MySQL
2589: - InteriorInstance
2730: ExamplePath - Path
2767: ExamplePath6 - Marker
2764: ExamplePath5 - Marker
2761: ExamplePath4 - Marker
2758: ExamplePath3 - Marker
2735: ExamplePath2 - Marker
2732: ExamplePath1 - Marker

Name: ExamplePath1

Transform

position	36.0809 -56.9544 1.4
rotation	0 0 1 239.496
scale	1 1 1

Misc

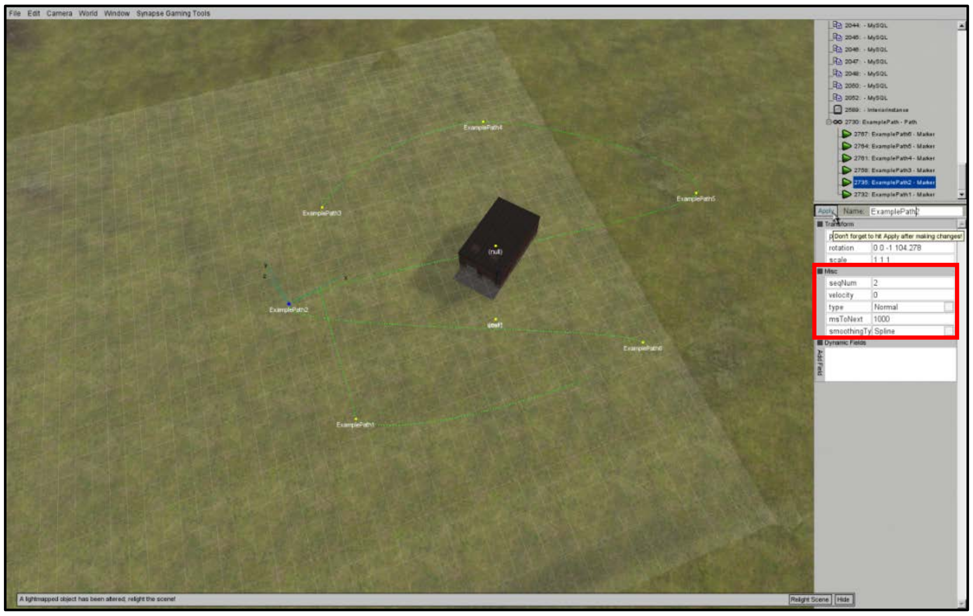
seqNum	6
velocity	0
type	Normal
msToNext	1000
smoothingTy	Spline

Dynamic Fields

A lightmapped object has been altered, relight the scene!

Fig. F-4 Linking the path

5. To ensure the vehicle will be traveling in the correct direction, check and fix the sequence order (seqNum) for each of the markers (Fig. F-5, red box). Click on the sequence number in the Window Editor Inspector (F3) menu, type in the correct number, and click Apply (e.g., the marker named ExamplePath1 should have a Sequence number of 1). During this process, it is normal for the green dotted line to look askew (Fig. F-5). It will become smooth again when all of the PathMarkers are set to the appropriate sequence and all of the Y-axes are facing the correct direction.


File Edit Camera World Window Synapse Gaming Tools
ExamplePath4
ExamplePath5
ExamplePath3
ExamplePath2
ExamplePath6
ExamplePath1
(null)
A lightmapped object has been altered; relight the scene!
Relight Scene
Hide

2044: - MySQL
2045: - MySQL
2046: - MySQL
2047: - MySQL
2048: - MySQL
2050: - MySQL
2052: - MySQL
2589: - InteriorInstance
2730: ExamplePath - Path
2767: ExamplePath6 - Marker
2764: ExamplePath5 - Marker
2761: ExamplePath4 - Marker
2758: ExamplePath3 - Marker
2735: ExamplePath2 - Marker
2732: ExamplePath1 - Marker

Apply Name: ExamplePath2

Transform
Don't forget to hit Apply after making changes!

rotation	0 0 -1 104.278
scale	1 1 1

Misc

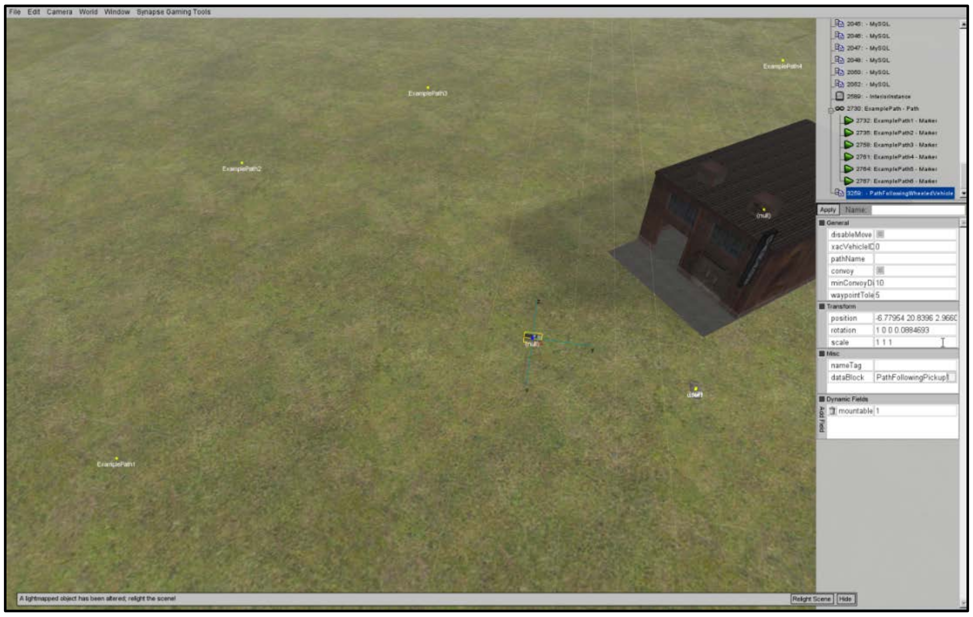
seqNum	2
velocity	0
type	Normal
msToNext	1000
smoothingTy	Spline

Dynamic Fields
Add Field

Fig. F-5 Setting the path sequence order (seqNum)

Appendix G. Add a Path-Following Vehicle

1. **Adding a path-following vehicle:** Open the World Editor Creator menu. Select Shapes, select Vehicles, and then choose one of the path-following vehicles (e.g., PathFollowingPickup1) (Fig. G-1). Highlight the vehicle and open the World Editor Inspector to set the criteria for the vehicle.



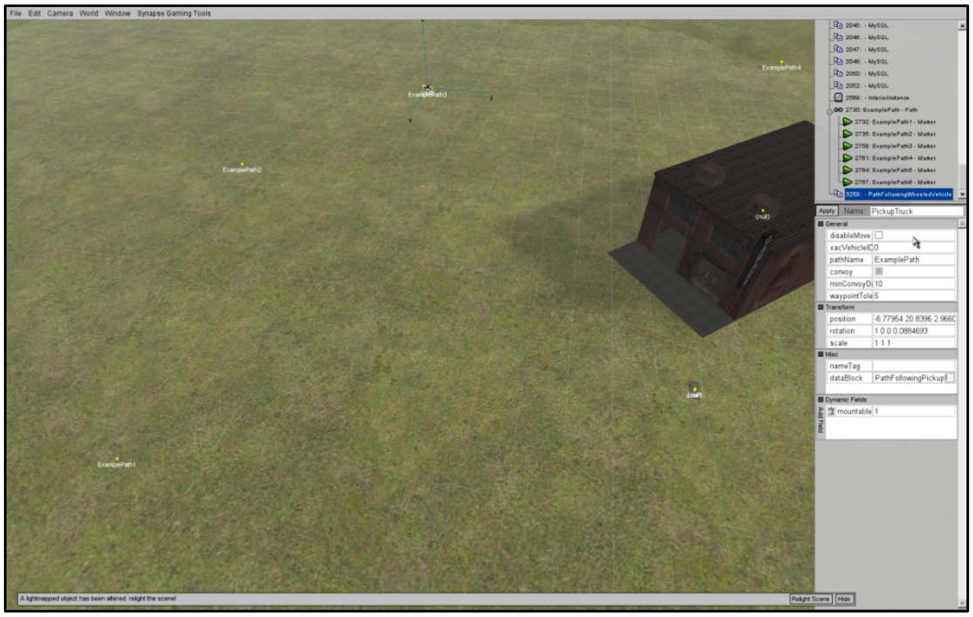
A lightmapped object has been altered; relight the scene!

Name:

General	
disableMove	☒
xacVehicleID	0
pathName	
convoy	☒
minConvoyDi	10
waypointTole	5
Transform	
position	-6.77954 20.8396 2.9660
rotation	1 0 0 0.0884693
scale	1 1 1
Misc	
nameTag	
dataBlock	PathFollowingPickup1
Dynamic Fields	
mountable	1

Fig. G-1 Path-following vehicle: setting vehicle criteria

2. **Adapting the criteria.** In the World Editor Inspector (F3), give the truck a name, enter the PathName, and disableMove (Fig. G-2).



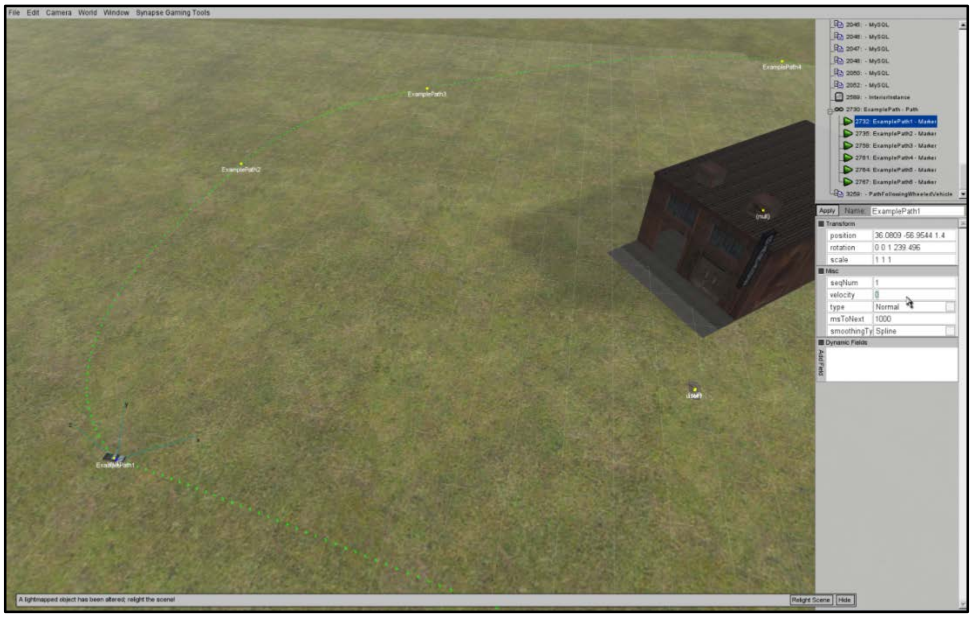
A lightmapped object has been altered; relight the scene!

Name: PickupTruck

General	
disableMove	☐
xacVehicleID	0
pathName	ExamplePath
convoy	☒
minConvoyDi	10
waypointTole	5
Transform	
position	-6.77954 20.8396 2.9660
rotation	1 0 0 0.0884693
scale	1 1 1
Misc	
nameTag	
dataBlock	PathFollowingPickup1
Dynamic Fields	
mountable	1

Fig. G-2 Path-following vehicle: adapting criteria

3. **Adjust the vehicle's speed.** Right now the truck bounces from marker to marker. This is happening because the truck's velocity is currently set to zero. To adjust the velocity (Fig. G-3), click on each marker and set a new speed (meters per second). Adjusting the speed of the vehicle at each marker allows for greater flexibility and control within the experiments.


File Edit Camera World Window Synapse Gaming Tools
ExamplePath4
ExamplePath3
ExamplePath2
ExamplePath1
(null)
2730: ExamplePath - Path
2732: ExamplePath1 - Marker
2735: ExamplePath2 - Marker
2758: ExamplePath3 - Marker
2761: ExamplePath4 - Marker
2764: ExamplePath5 - Marker
2767: ExamplePath6 - Marker
3259: - PathFollowingWheeledVehicle
Apply
Transform
Misc
Dynamic Fields
Add Field
Relight Scene
Hide

Name: ExamplePath1

position	36.0809 -56.9544 1.4
rotation	0 0 1 239.496
scale	1 1 1
seqNum	1
velocity	0
type	Normal
msToNext	1000
smoothingTy	Spline

A lightmapped object has been altered; relight the scene!

Fig. G-3 Path-following vehicle: adjusting the vehicle speed

INTENTIONALLY LEFT BLANK.

Appendix H. Trigger a Non-player Character

This appendix appears in its original form, without editorial change.

It is possible to add a non-player character (NPC) to this Path starting at PathMarker1 at a specific time. To accomplish this task requires some programming and creation of three new TorqueScript files, which we named HRIexperimentAIplayer.cs, spawnNPC.cs, and triggerNPC.cs. The code samples included below can be placed directly into your copy of RIVET. Here, we provide the TorqueScript to include SoldierPlayer1 (the animations for this NPC have already been included in a previous version of RIVET and can be located in sim.base/data/shapes/soldier2 for the Soldiers, sim.base/data/shapes/terrorists for insurgents, and sim.base/data/shapes/player for the civilians).

1. **Initialize the NPCs (Soldier Player 1):** Create a new TorqueScript file. Open a text file and add the following code block. This file will link to the people animations already available in RIVET. Save as a unique file name (e.g., HRIexperimentAIplayer.cs) in the following directory C:\RIVET\sim.base\server. This file initializes the NPC and allows them to be spawned on a path and to move on a path. For a reference point also look at a previously created Player file (simbase/server/aiPlayer.cs). Green text represents comments in the TorqueScript code. All comments are also delineated with two forward slash lines at the beginning of the line.

```
//-----------------------------------------------------------------------------
// Initializing NPCs
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// AIPlayer callbacks
// The AIPlayer class implements the following callbacks:
//
//    PlayerData::onStuck(%this,%obj)
//    PlayerData::onUnStuck(%this,%obj)
//    PlayerData::onStop(%this,%obj)
//    PlayerData::onMove(%this,%obj)
//    PlayerData::onReachDestination(%this,%obj)
//    PlayerData::onTargetEnterLOS(%this,%obj)
//    PlayerData::onTargetExitLOS(%this,%obj)
//    PlayerData::onAdd(%this,%obj)
//
// Since the AIPlayer doesn't implement its own datablock, these callbacks
// all take place in the PlayerData namespace.
//-----------------------------------------------------------------------------
function BasePlayer::onReachDestination(%this,%obj){
    if (%obj.crowd !$= "") {
        %endPath = %obj.crowd.isEndOfPath(%obj.index);
         if(%endPath)     {
                AIPlayer::decideAction(%obj);
        }
         else {
                AIPlayer::nextPathDestination(%obj);
        }
    }
     else   {
        //echo( "AIPlayer::onReachDestination warning - Crowd is blank!" );
    }
    if (%obj.path !$= "") {
        if (%obj.currentNode == %obj.targetNode)
        %this.onEndOfPath(%obj,%obj.path);
```

```
          else
        %obj.moveToNextNode();
    }
    else   {
                //echo( "AIPlayer::onReachDestination warning - Path is blank!" );
    }
}

function BasePlayer::onIdleOver(%this,%obj) {
     if (%obj.crowd !$= "")   {
                AIPlayer::decideAction(%obj);
    }
    else   {
        //echo( "AIPlayer::onIdleOver warning - Crowd is blank!" );
    }
}

function BasePlayer::onMoveStuck(%this,%obj){
    if (%obj.crowd !$= "")   {
        AIPlayer::stuck(%obj);
    }
     else   {
        //echo( "AIPlayer::onMoveStuck warning - Crowd is blank!" );
    }
}

function BasePlayer::onEndOfPath(%this,%obj,%path) {
    %obj.nextTask();
    // %obj.moveToNode(0);
}

function BasePlayer::onEndSequence(%this,%obj,%slot) {
    echo("Sequence Done!");
    %obj.stopThread(%slot);
    %obj.nextTask();
}

//-----------------------------------------------------------------------------
// SoldierPlayer1
// Add this section for all NPCs that you have animations available
//-----------------------------------------------------------------------------

datablock PlayerData(SoldierPlayer1 : SoldierBodyType1) {
    shootingDelay = 2000;
};

function SoldierPlayer1::onReachDestination(%this,%obj) {
     BasePlayer::onReachDestination(%this,%obj);
}

function SoldierPlayer1::onIdleOver(%this,%obj) {
    BasePlayer::onIdleOver(%this,%obj);
}

function SoldierPlayer1::onMoveStuck(%this,%obj) {
    BasePlayer::onMoveStuck(%this,%obj);
}

function SoldierPlayer1::onEndOfPath(%this,%obj,%path) {
    BasePlayer::onEndOfPath(%this,%obj,%path);
}

function SoldierPlayer1::onEndSequence(%this,%obj,%slot) {
    BasePlayer::onEndSequence(%this,%obj,%slot);
}
```

```
//-----------------------------------------------------------------------------
// AIPlayer static functions
//-----------------------------------------------------------------------------

function AIPlayer::spawn(%datablock, %name, %spawnPoint)
{
   %player = new AIPlayer(){
       dataBlock = %datablock;
      path = "";
   };
   echo("AIPlayer::spawn - Player:",%player);
   MissionCleanup.add(%player);
   %player.setShapeName(%name);
   %player.setName( %name );
   %player.setTransform(%spawnPoint);
   echo("AIPlayer::spawn - spawnPoint:",%spawnPoint);

   return %player;
}

function AIPlayer::spawnOnPath(%datablock, %name, %path) {
    // Spawn a player and place him on the first node of the path
    echo("AIPlayer::spawnOnPath - Path:",%path);
    if (!isObject(%path)) {
          echo("AIPlayer::spawnOnPath - Cannot Find Path!");
           return;
   }
   %node = %path.getObject(0);
   echo("AIPlayer::spawnOnPath - Node:",%node);
   echo("AIPlayer::spawnOnPath - Node Transform:",%node.getTransform());
    %player = AIPlayer::spawn(%datablock, %name,%node.getTransform());
    echo("AIPlayer::spawnOnPath - Player:",%player);

    return %player;
}

//-----------------------------------------------------------------------------
// AIPlayer methods
//-----------------------------------------------------------------------------

function AIPlayer::followPath(%this,%path,%node) {
   // Start the player following a path
   echo("AIPlayer::followPath - Entered method.\n");

   %this.stopThread(0);

   if (!isObject(%path)) {
      %this.path = "";
      echo("AIPlayer::followPath - Attempting to follow bad path.\n");
       return;
   }

   if ((%node > %path.getCount() - 1))
      %this.targetNode = %path.getCount() - 1;
   else
      %this.targetNode = %node;

   if (%this.path $= %path)
      %this.moveToNode(%this.currentNode);
   else {
      %this.path = %path;
      %this.moveToNode(0);
       //%this.currentNode = 0;
   }
}
```

```
function AIPlayer::moveToNextNode(%this)
{
    if (%this.targetNode < 0 || %this.currentNode < %this.targetNode) {
        if (%this.currentNode < %this.path.getCount() - 1)
            %this.moveToNode(%this.currentNode + 1);
         else
            %this.moveToNode(0);
    }
    else
        if (%this.currentNode == 0)
            %this.moveToNode(%this.path.getCount() - 1);
         else
            %this.moveToNode(%this.currentNode - 1);
}

function AIPlayer::moveToNode(%this,%index)
{
    // Move to the given path node index
    %this.currentNode = %index;
    %node = %this.path.getObject(%index);
    %this.setMoveDestination(%node.getTransform(), %index == %this.targetNode);
}


function AIPlayer::pushTask(%this,%method) {
    if (%this.taskIndex $= ""){
        %this.taskIndex = 0;
        %this.taskCurrent = -1;
    }
    %this.task[%this.taskIndex] = %method;
    %this.taskIndex++;
    if (%this.taskCurrent == -1)
        %this.executeTask(%this.taskIndex - 1);
}

function AIPlayer::clearTasks(%this) {
    %this.taskIndex = 0;
    %this.taskCurrent = -1;
}

function AIPlayer::nextTask(%this) {
    if (%this.taskCurrent != -1)
        if (%this.taskCurrent < %this.taskIndex - 1)
                %this.executeTask(%this.taskCurrent++);
        else
                %this.taskCurrent = -1;
}

function AIPlayer::executeTask(%this,%index) {
    %this.taskCurrent = %index;
    eval(%this.getId() @ "." @ %this.task[%index] @ ";");
}

//-----------------------------------------------------------------------------
function AIPlayer::singleShot(%this)
{
    // The shooting delay is used to pulse the trigger
    %this.setImageTrigger(0,true);
    %this.setImageTrigger(0,false);
    %this.trigger = %this.schedule(%this.shootingDelay,singleShot);
}
//-----------------------------------------------------------------------------

function AIPlayer::wait(%this,%time) {
    %this.schedule(%time * 1000,"nextTask");
}

function AIPlayer::done(%this,%time) {
    %this.schedule(0,"delete");
}
```

```
function AIPlayer::fire(%this,%bool) {
    if (%bool) {
         cancel(%this.trigger);
        %this.singleShot();
    }
    else
        cancel(%this.trigger);
        %this.nextTask();
}

function AIPlayer::aimAt(%this,%object){
    echo("Aim: " @ %object);
    %this.setAimObject(%object);
    %this.nextTask();
}

function AIPlayer::animate(%this,%seq) {
    echo("Set animation: " @ %seq);
    //%this.stopThread(0);
    //%this.playThread(0,%seq);
    %this.setActionThread(%seq);
}
return %player;
```

2. In the same directory, open game.cs and add an executable (exec) function to this file around lines 190-200 exec("./HRIexperimentAIplayer.cs"); This will allow the new file to be executed during the Mission run. Please note that the exec function is colored purple. If you use Torsion IDE for developing code, all exec functions are purple (refer to pg. 38 for more information on Torsion).

3. Open the Mission file HRIExperiment1.mis (located in C:\RIVET\sim.base\data\user_missions) and update the initial mission info: new ScriptObject(MissionInfo). This allows the Mission File to call to the AIplayer information.

```
  new ScriptObject(MissionInfo) {
      name = "RIVET TR Example: HRI Experiment 1";
      desc0 = "Designed by Dr. K.E. Schaefer, ARL";
      aiPlayerFile = "HRIexperimentAIplayer.cs";
      descLines = "1";
      map = "ARL_world";
};
```

4. **Add TorqueScript File to Spawn a NPC:** Add the following code block into a text file and save as spawnNPC.cs in the following directory C:\RIVET\sim.base\server. This will allow NPCs to be created by a trigger file.

```
$tickNumber = 0;
$TotalExposureTime = 0;

//Hook into the mission editor ---------------------------------------------------------------------------
function spawnNPCTrigger::create(%data) {
   echo("spawnNPCTrigger::create");
   // The mission editor invokes this method when it wants to create an object of the given datablock type.
   %obj = new spawnNPCTrigger() {
   dataBlock = %data;
   };
return %obj;
}

// dummy datablock needed to create a trigger
// the values in this datablock are not used

datablock TriggerData(spawnNPCTrigger) {
    tickPeriodMS = 100;
};

//Events ------------------------------------------------------------------------
//this event is fired from the mission editor when a spawnNPCTrigger is added

function spawnNPCTrigger::onAdd(%this,%obj) //this event is fired from the mission editor
{
   Parent::onAdd(%this,%obj);
   echo("spawnSoldierTrigger::onAdd");
   %size = %this.triggerRadius;
   %scale = VectorScale(%obj.scale,%size*2);
   %sizeVec = %size @" "@ %size @" "@ %size;
   %pos = VectorAdd(%obj.position,%sizeVec);
   // now we have to give the static shape a controlling trigger of the right size and location
   %trigger = new Trigger() {
      dataBlock = spawnSoldierTrigger;
         position = %pos;
         rotation = %obj.rotation;
         scale = %scale;
         polyhedron = "0.000000 1.0000000 0.0000000 1.0000000 0.0000000 0.0000000 0.0000000 -1.0000000
         0.0000000 0.0000000 0.0000000 1.0000000";
      };
   %trigger.setActive(true);
   %trigger.setOwner(%obj);
   %obj.setTrigger(%trigger);
}

//this event is fired when an object enters the trigger

function spawnNPCTrigger::onEnterTrigger(%this,%trigger,%obj)
{
   if(strcmp(%obj.getName(), "user1") == 0)   {
      echo("spawnNPCTrigger::onEnterTrigger spawning " @ %trigger.npcType @ " on path " @
%trigger.npcPath);
      echo("Vehicle name " @ %obj.getName() );
      %player = AIPlayer::spawnOnPath(%trigger.npcType,"SoldierNPC",%trigger.npcPath);
      %player.followPath(%trigger.npcPath,-1);
      %player.setMoveSpeed(0.4);//m/s
      // %player.setInventory(M4,1);
      // %player.setInventory(M4Ammo,100);
      // %player.mountImage(M4Image,0);
   }
}
```

5. In the same directory, open game.cs and add an exec function to this file around lines 190-200 exec("./spawnNPC.cs"); This will allow the new file to be accessed during the Mission run.

6. **Add a Trigger to the Mission File:** A trigger will be represented by a pink box in the VE. When your Player Character (e.g., Soldier or vehicle) crosses through the Trigger point, it will initiate the Spawn function of the NPC at Path Marker 1. Open Mission File and add this code.

```
new Trigger(NPC1) {
   position = "-12.899 44.4245 0.8";
   rotation = "0 0 1 213.805";
   scale = "10 3 2";
   dataBlock = "spawnNPCTrigger";
   polyhedron = "0.0000000 0.0000000 0.0000000 1.0000000 0.0000000 0.0000000 0.0000000 -1.0000000
   0.0000000 0.0000000 0.0000000 1.0000000";
   onGroup = "Default Value";
   npcType = "SoldierPlayer1";
   npcPath = "MissionGroup/ExamplePath";
};
```

NPC1: This is the name we provided to this Non-Player Character. You can edit this to be a unique name.

Position: This is the location of where the Trigger will be located in the VE. A good starting point is either near the initial SpawnSphere or near your PathMarker1. You will be able to move it to wherever you need when you open the VE.

Rotation: This is the orientation of the Trigger. It is adaptable within the VE.

Scale: This is the size of the Trigger. It is adaptable.

dataBlock: This is what calls to the spawnNPC.cs file to create the NPC.

npcType: This tells the spawnNPC.cs which character you want to create. **If you directly copy these TorqueScript files into your version of RIVET, you will only have access to SoldierPlayer1.

npcPath: This lets you set which path you want your NPC Soldier to be located. It is adaptable.

** If you want to trigger more Soldiers to this same path, all you need to do is add the code from Step #6 to the Mission file and move the location of the Trigger to meet your needs for the experimental Design.

Appendix I. Delete a Non-player Character with a Trigger

1. **Delete a NPC:** There may be times that you want to delete the NPC. Add the following code to the SpawnNPC.cs file

```
//-----------------------------------------------
//Delete NPC at a Trigger
//----------------------------------------------

datablock TriggerData(deleteNPCTrigger)
{
   tickPeriodMS = 100;
};

function deleteNPCTrigger::onEnterTrigger(%this,%trigger,%obj)
{
   if(isObject(SoldierNPC) || isObject(Civilian))
   {
      echo("deleteNPCTrigger::onEnterTrigger deleting " @ %obj.getname());
      %obj.applyDamage(10000);
      // %obj.delete();
   }
}
```

2. **Add a Delete Trigger to the VE:** Add the following code to the HRIExperiment1.mis file to create the Delete NPC trigger

```
new Trigger(delete_NPC1) {
   position = "-36.6411 -71.5004 0.4";
   rotation = "0 0 -1 88.8997";
   scale = "5 3 2";
   dataBlock = "deleteNPCTrigger";
   polyhedron = "0.0000000 0.0000000 0.0000000 1.0000000 0.0000000 0.0000000 0.0000000 -1.0000000
   0.0000000 0.0000000 0.0000000 1.0000000";
};
```

Appendix J. Creating Crowds

1. **Open the Crowd Editor.** Double-click the CrowdEditor.exe file located in C:\RIVET\support\Crowd Tool.

2. **Load the Map.** The map is loaded by selecting File from the menu in the upper-left-hand corner of the application and choosing Load Map (Fig. J-1). This will bring up a dialog box to select the map file from the computer.

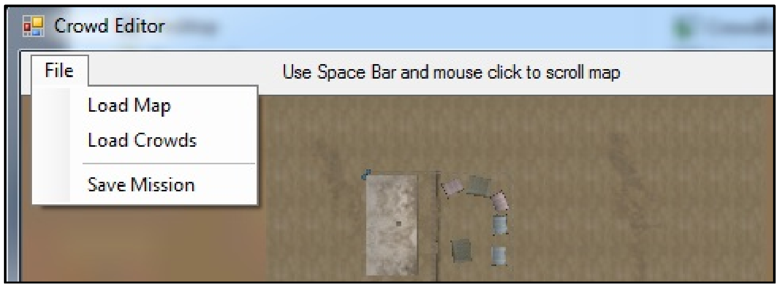


Fig. J-1 Crowd Editor menu

3. **Add a Crowd.** Click Add in the upper-right corner (Fig. J-2). Another GUI will open where the name of the crowd is entered. Currently, for the software to work properly, the crowds must be named Crowd1, Crowd2, … Crowd*n*, where n is the number for the last Crowd in the scene. This will generate information in the crowd list.

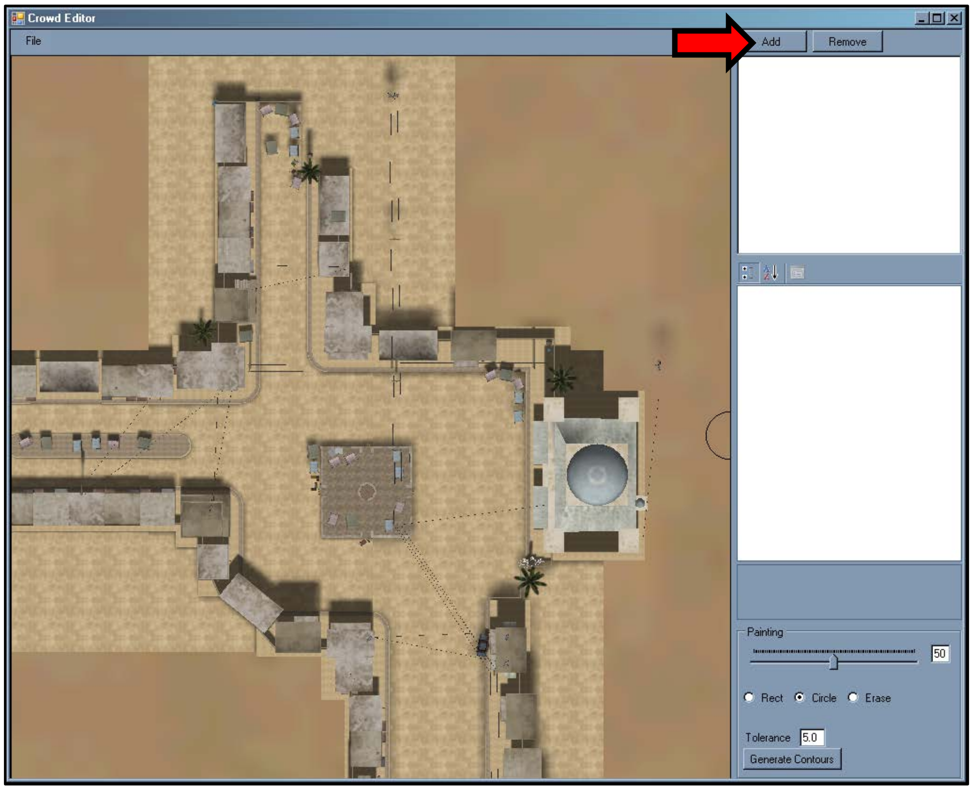
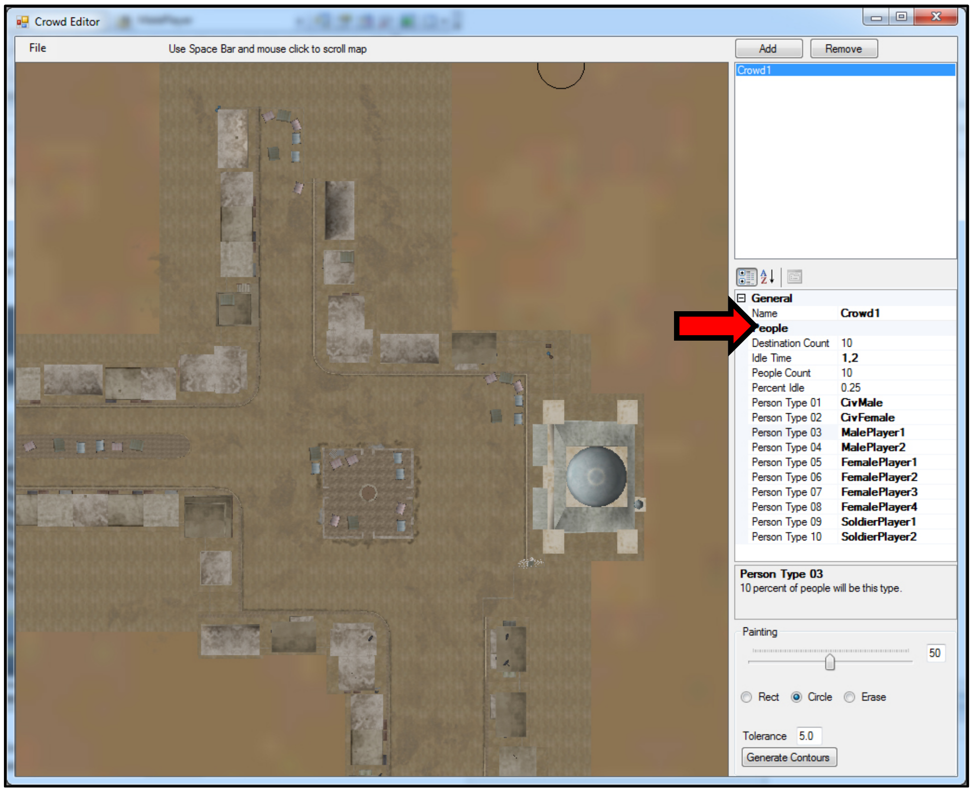


Fig. J-2 Add a crowd

4. **Adjust the Crowd's Properties.** The default settings will be used in the properties menu unless the destination count, people count, and percent idle are changed. The crowds are grouped in tens and each person type can be selected out of the possible choices (Fig. J-3).



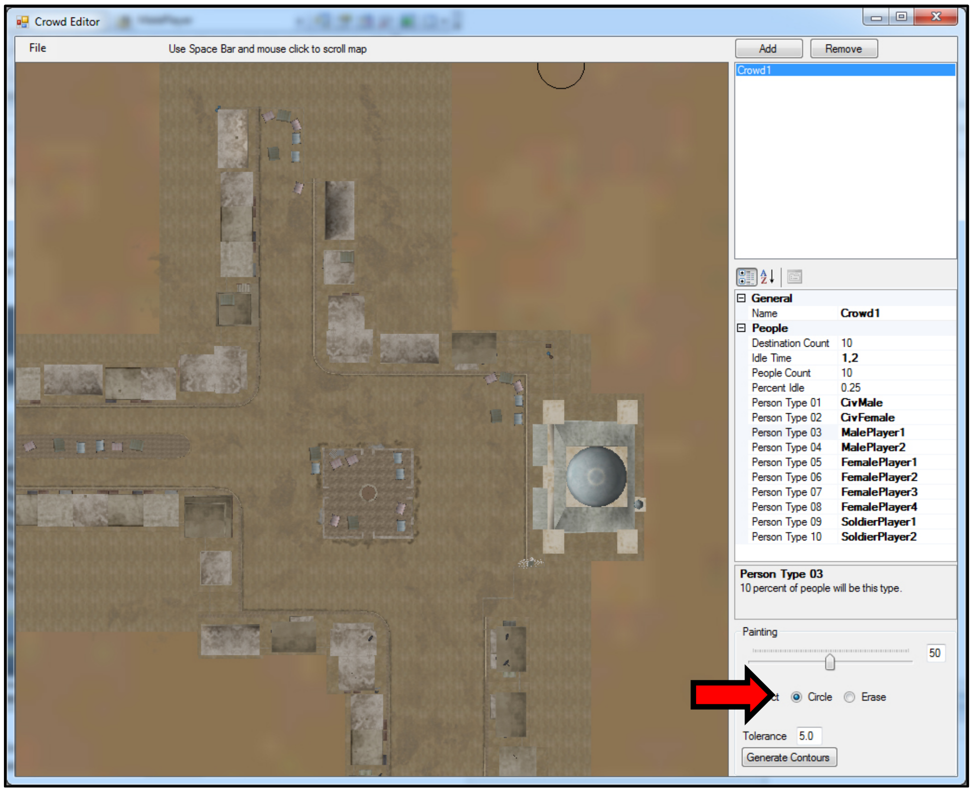
General	
Name	Crowd1
People	
Destination Count	10
Idle Time	1,2
People Count	10
Percent Idle	0.25
Person Type 01	CivMale
Person Type 02	CivFemale
Person Type 03	MalePlayer1
Person Type 04	MalePlayer2
Person Type 05	FemalePlayer1
Person Type 06	FemalePlayer2
Person Type 07	FemalePlayer3
Person Type 08	FemalePlayer4
Person Type 09	SoldierPlayer1
Person Type 10	SoldierPlayer2

Person Type 03
10 percent of people will be this type.

Painting: 50
Rect [✓ Circle] Erase
Tolerance 5.0

Fig. J-3 Crowd properties menu

5. **Define the Location for the Crowd.** Choose the size of the brush using the Painting slider (Figs. J-4 and J-5). This will adjust the area of the brush. The outline choice is either a rectangle or a circle, which is the default. The Erase Radio button is used to undo any painting previously done.



General	
Name	Crowd1
People	
Destination Count	10
Idle Time	1,2
People Count	10
Percent Idle	0.25
Person Type 01	CivMale
Person Type 02	CivFemale
Person Type 03	MalePlayer1
Person Type 04	MalePlayer2
Person Type 05	FemalePlayer1
Person Type 06	FemalePlayer2
Person Type 07	FemalePlayer3
Person Type 08	FemalePlayer4
Person Type 09	SoldierPlayer1
Person Type 10	SoldierPlayer2

Person Type 03
10 percent of people will be this type.

Painting 50

[✓ Circle] Erase

Tolerance 5.0

Generate Contours

Fig. J-4 Crowd Location menu

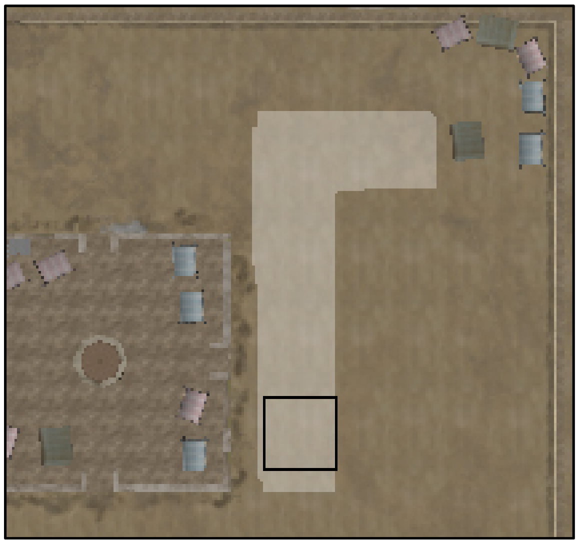
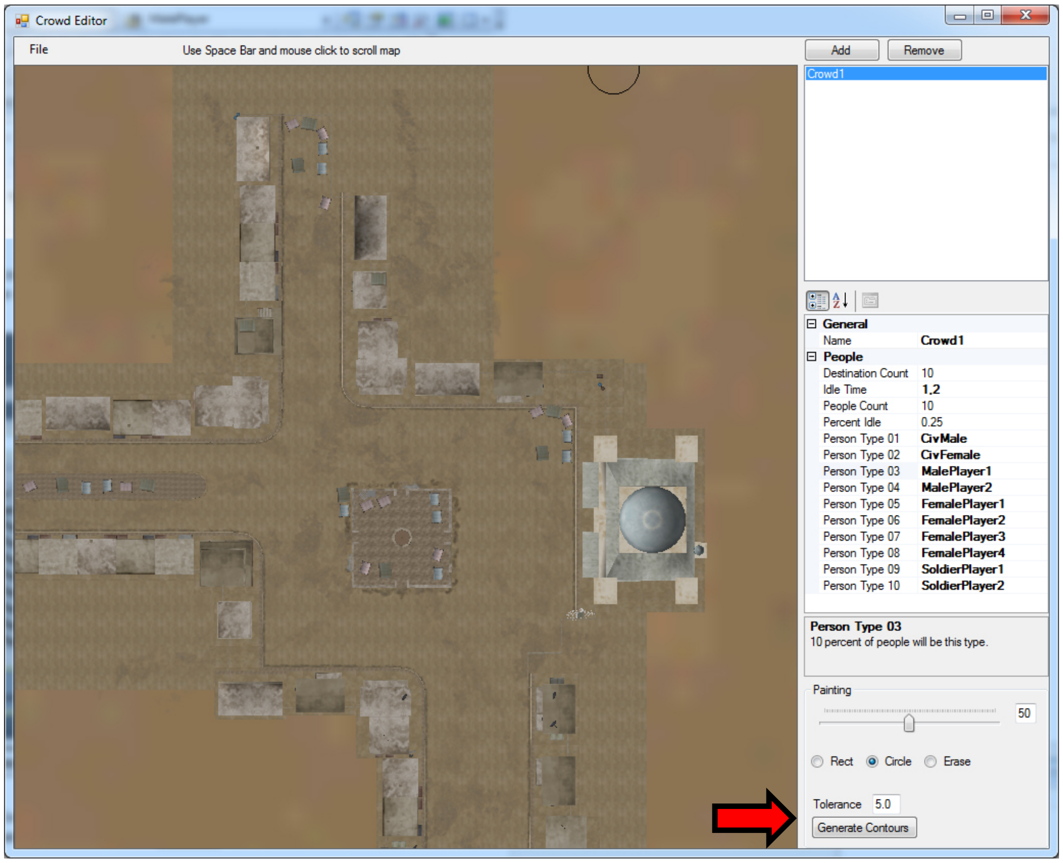


Fig. J-5 Crowd location area

6. **Generate the Contour.** The tolerance for the contour generation is set to 5.0. By clicking on Generate Contours, the application will outline the area set for the crowd (Figs. J-6 and J-7).



General

Name	Crowd1

People

Destination Count	10
Idle Time	1.2
People Count	10
Percent Idle	0.25
Person Type 01	CivMale
Person Type 02	CivFemale
Person Type 03	MalePlayer1
Person Type 04	MalePlayer2
Person Type 05	FemalePlayer1
Person Type 06	FemalePlayer2
Person Type 07	FemalePlayer3
Person Type 08	FemalePlayer4
Person Type 09	SoldierPlayer1
Person Type 10	SoldierPlayer2

Person Type 03
10 percent of people will be this type.

Painting 50

( ) Rect (•) Circle ( ) Erase

Tolerance 5.0

Fig. J-6 Generate Contour menu

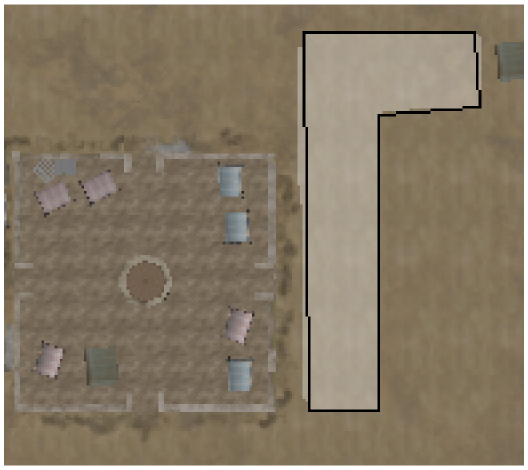


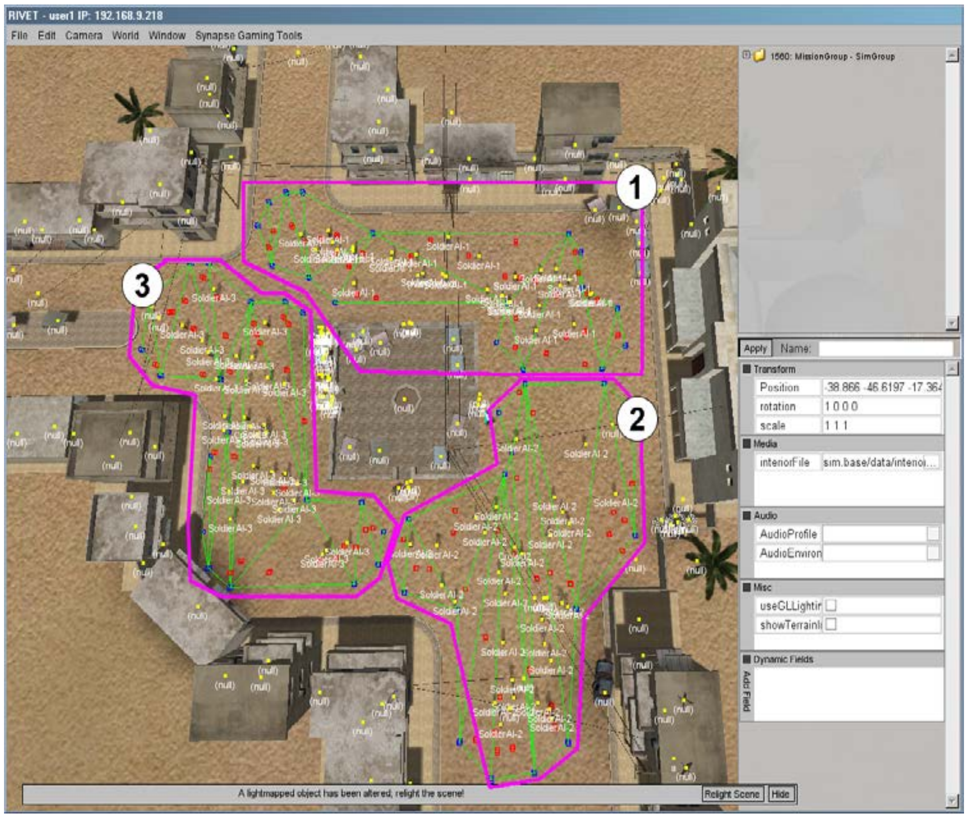
Fig. J-7 Contour area

7. **Save Mission.** Choose Save Mission from the menu and then find the mission file (.mis) with which this crowd will be associated. It will ask, “Do you want to overwrite?” Click yes and it will add the following to the end of the TorqueScript file. These are the parameters set in the GUI. The positions are the corners of the boundary line around the contour, which will enable RIVET to create the crowd area inside the application.

```
new Crowd(Crowd1) {
    PeopleCount = "10";
    DestinationCount = "10";
    IdlePercent = "0.25";
    IdleTime = "1 2";
    AvailablePerson[0] = "CivMale";
    AvailablePerson[1] = "CivFemale";
    AvailablePerson[2] = "MalePlayer1";
    AvailablePerson[3] = "MalePlayer2";
    AvailablePerson[4] = "FemalePlayer1";
    AvailablePerson[5] = "FemalePlayer2";
    AvailablePerson[6] = "FemalePlayer3";
    AvailablePerson[7] = "FemalePlayer4";
    AvailablePerson[8] = "SoldierPlayer1";
    AvailablePerson[9] = "SoldierPlayer2";
    Positions = "10.8 3.2,11.2 -25.2,16.4 -25.2,16.4 -3,24 -2.4,23.6 3.2";
  };
```

Once a mission is complete, there is always an opportunity to make changes by choosing File, Load Crowds from the main menu and selecting the mission to change. This selection will then read in the crowd details in the GUI and allow changes to any part of the crowd. Ensure that Generate Contours is selected before saving if the contour is repainted.

8. **Example.** An example of the way to script a file for crowds is in the crowdAIplayer.cs file. Once the mission is launched, the crowds will appear. An overhead view from the World Editor in Fig. J-8 shows 3 crowds labeled 1, 2, and 3. The blue squares are the contour points noted in the mission file. The green lines are contour boundaries while the red squares are the destination points to which the crowd participants will travel. These points are chosen randomly for each AI player entity.



Position -38.866 -46.6197 -17.364
rotation 1 0 0 0
scale 1 1 1
interiorFile sim.base/data/interio...
AudioProfile
AudioEnviron
useGLLightin
showTerrainli

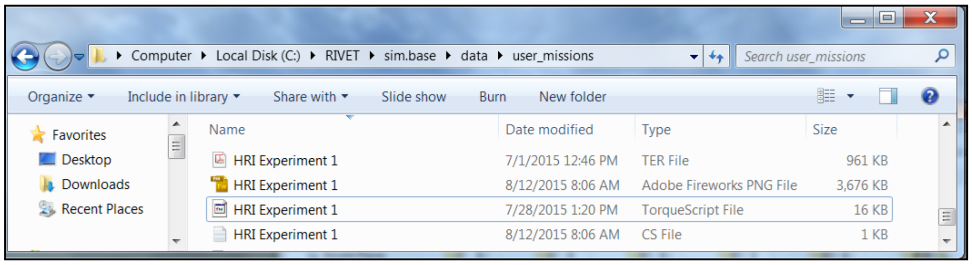
A lightmapped object has been altered; relight the scene!

Fig. J-8 Overhead view of multiple crowds in World Editor

INTENTIONALLY LEFT BLANK.

Appendix K. Loading the CARVE Application

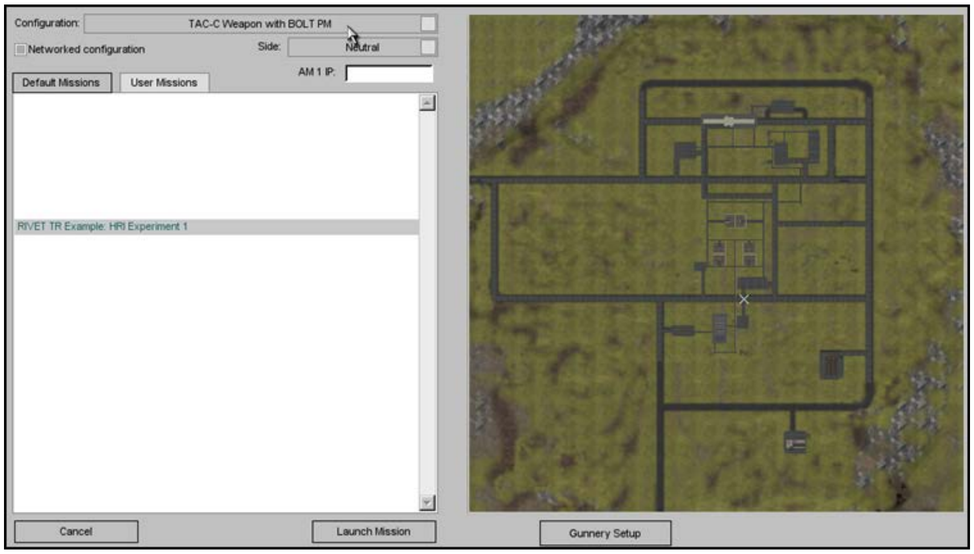
1. **Setup.** Prior to opening CARVE, check to make sure that both the RIVET Server computer, and the RIVET BOLT computer user mission directories include the Mission File (.mis), Terrain File (.ter), Map (.png), and associated .cs file. Specific directions on how to build these files are in Appendix B.



Name	Date modified	Type	Size
HRI Experiment 1	7/1/2015 12:46 PM	TER File	961 KB
HRI Experiment 1	8/12/2015 8:06 AM	Adobe Fireworks PNG File	3,676 KB
HRI Experiment 1	7/28/2015 1:20 PM	TorqueScript File	16 KB
HRI Experiment 1	8/12/2015 8:06 AM	CS File	1 KB

Fig. K-1 Required files

2. **Start RIVET Server.** Click on RIVET shortcut. Set vehicle to be a BOLT-enabled vehicle, select the User Mission, and click Launch Mission (Fig. K-2).



Configuration: TAC-C Weapon with BOLT PM

Networked configuration

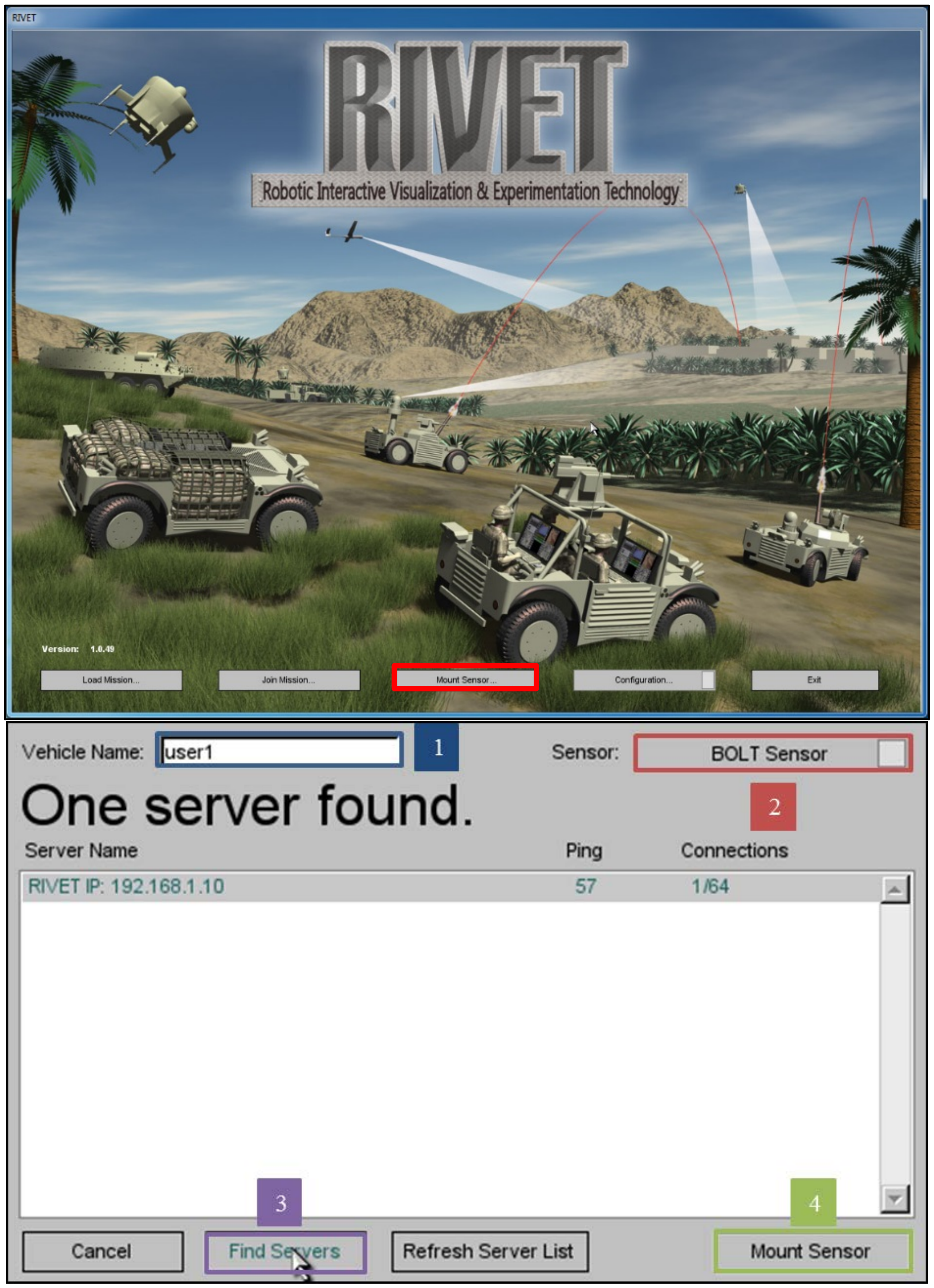
Side: Neutral

AM 1 IP:

RIVET TR Example: HRI Experiment 1

Fig. K-2 Start RIVET Server computer with BOLT-enabled vehicle

3. **Start RIVET Client.** Mount the BOLT sensor. This option allows users to BOLT an additional sensor on the vehicle and is required if the user will be using the CARVE interface. After selecting the Mount Sensor button, a new GUI menu will open. At this time, check to make sure the vehicle name is set to user 1 [1], and that the sensor selected is the BOLT sensor [2]. Next, click on Find Servers [3], select the correct IP, and select Mount Sensor [4] (Fig. K-3).



Vehicle Name: user1

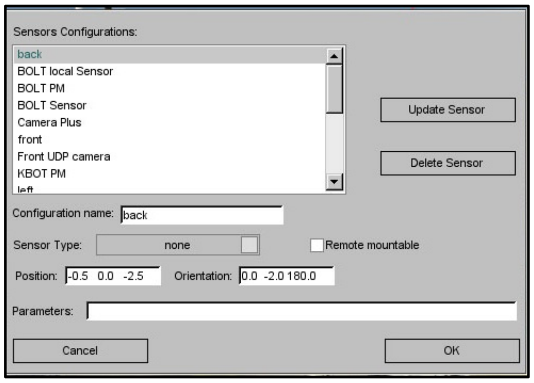
Sensor: BOLT Sensor

One server found.

Server Name	Ping	Connections
RIVET IP: 192.168.1.10	57	1/64

Fig. K-3 Mount the RIVET BOLT sensor to the vehicle using the Client computer

It is also possible to customize the sensors through the sensor drop-down menu (Fig. K-3, [2]).Vehicles are equipped with various sensors that are configurable through the Sensor Configuration GUI (Fig. K-4).



back
BOLT local Sensor
BOLT PM
BOLT Sensor
Camera Plus
front
Front UDP camera
KBOT PM

Configuration name: back

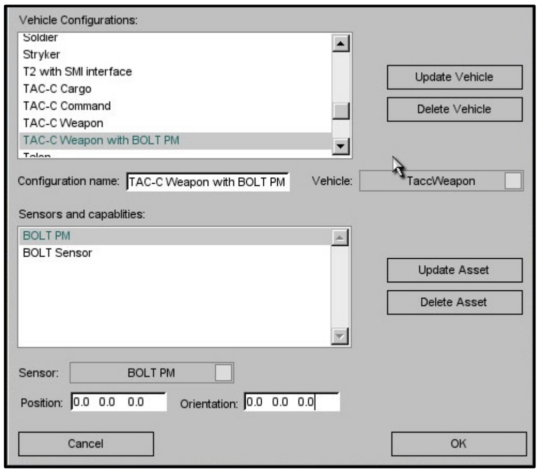
Sensor Type: none

Position: -0.5 0.0 -2.5 Orientation: 0.0 -2.0 180.0

Parameters:

Fig. K-4 Sensor Configuration GUI

The vehicles are selected in the drop-down box (Fig. K-5). These are configurable prior to the load mission menu using vehicle configuration GUI. Sensors that were configured are added to each vehicle configuration with a given position and orientation.



Soldier
Stryker
T2 with SMI interface
TAC-C Cargo
TAC-C Command
TAC-C Weapon
TAC-C Weapon with BOLT PM

Configuration name: TAC-C Weapon with BOLT PM Vehicle: TaccWeapon

BOLT PM
BOLT Sensor

Sensor: BOLT PM

Position: 0.0 0.0 0.0 Orientation: 0.0 0.0 0.0

Fig. K-5 Vehicle configuration

Sensors can be local to the server or loaded on the Client machine and added through the Mount Sensor GUI (Fig. K-6).

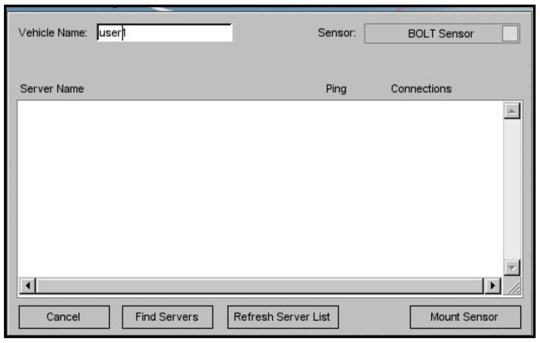


Fig. K-6 Mount Sensor GUI

4. **Open CARVE.** Now that both the RIVET server and RIVET client BOLT sensor computers are activated, click on the CARVE shortcut icon (Fig. K-7).

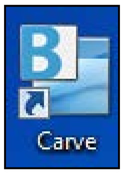


Fig. K-7 CARVE shortcut icon

5. **Check Connectivity.** Check connectivity to make sure that CARVE can "talk" to the RIVET Server and Client machines (Figs. K-8 and K-9).

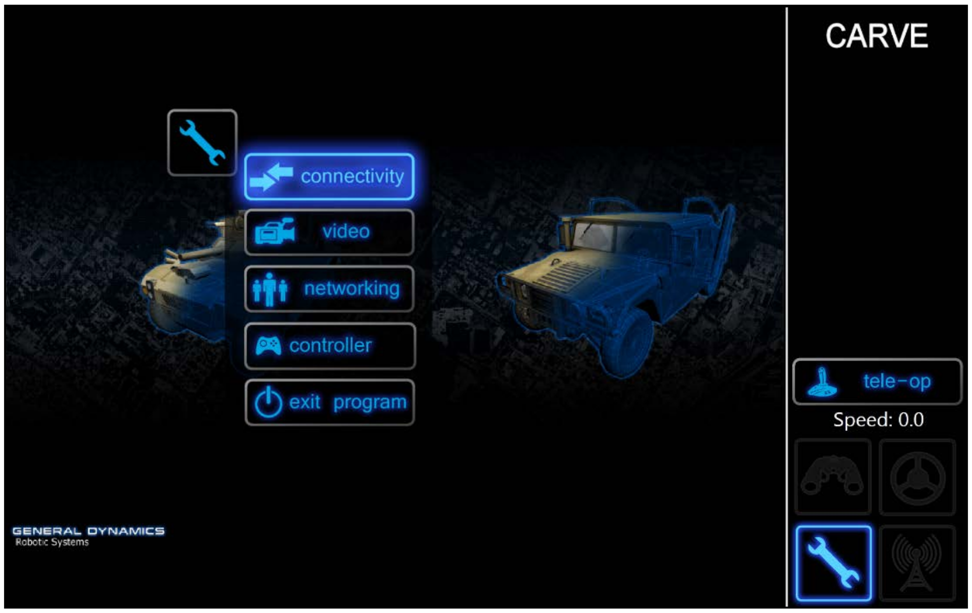
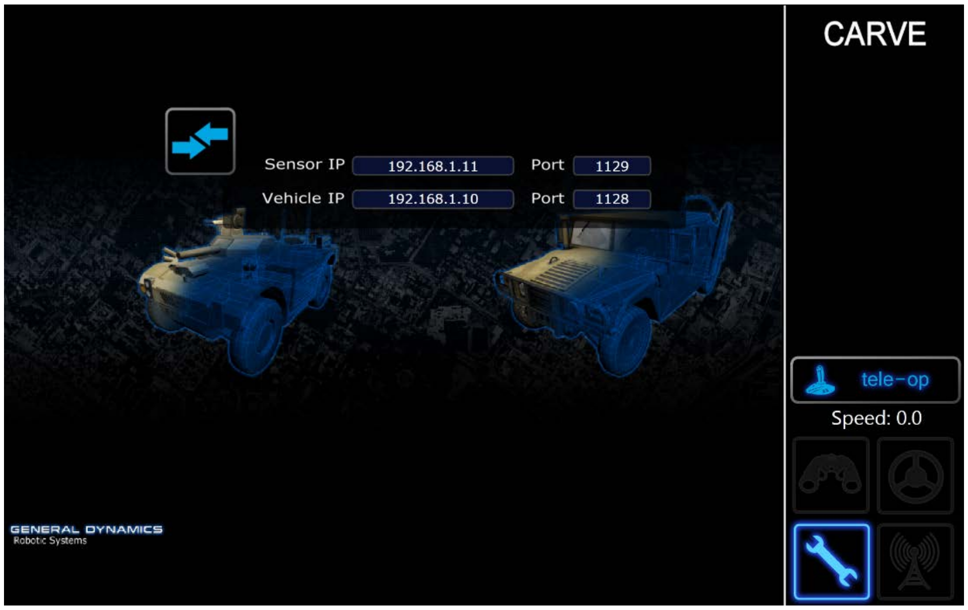


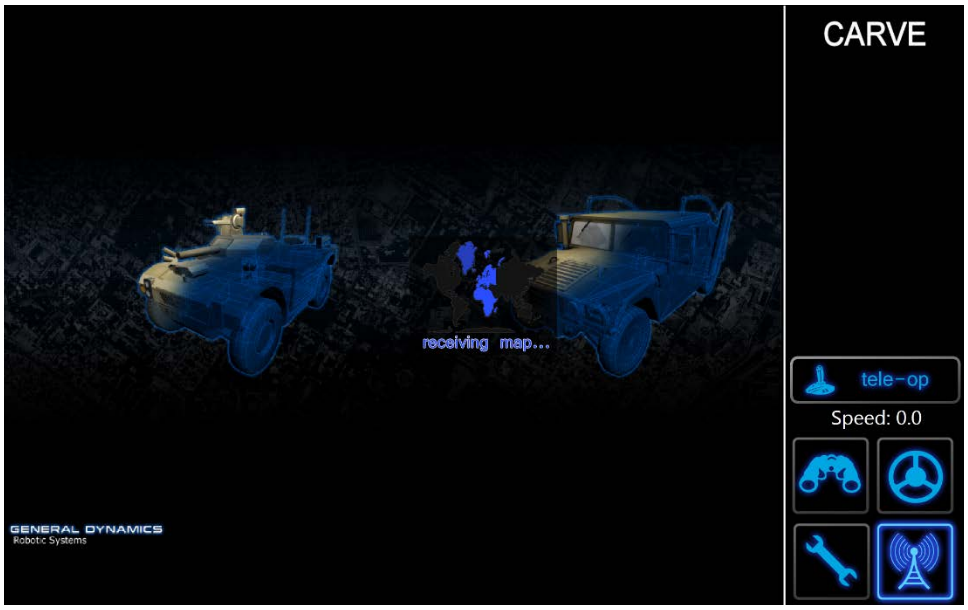
Fig. K-8 CARVE Connectivity tool



Sensor IP 192.168.1.11 Port 1129
Vehicle IP 192.168.1.10 Port 1128
Speed: 0.0

Fig. K-9 Check connectivity with RIVET machines

6. **Connect CARVE to RIVET.** During this initialization phase, CARVE will be receiving the map from RIVET (Fig. K-10). This can take upwards of a minute.



receiving map...
Speed: 0.0

Fig. K-10 Map upload from RIVET to CARVE

7. **Failure to Open.** If CARVE is unable to receive the map, it will provide an error message. At this point, follow the directions in Section 6.1 to identify the issue. If CARVE successfully loaded, the video sensor will appear in the main screen window.

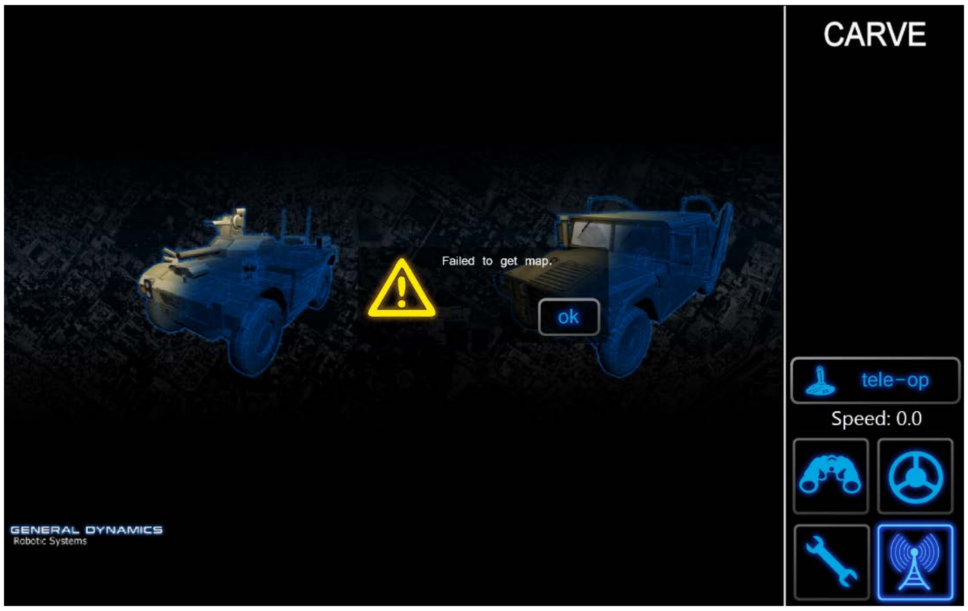


Fig. K-11 Example of the CARVE failure to load map screen

8. **Saving the Map.** For faster future loading, it is important to save the Map. First, open the map (Fig. K-12).

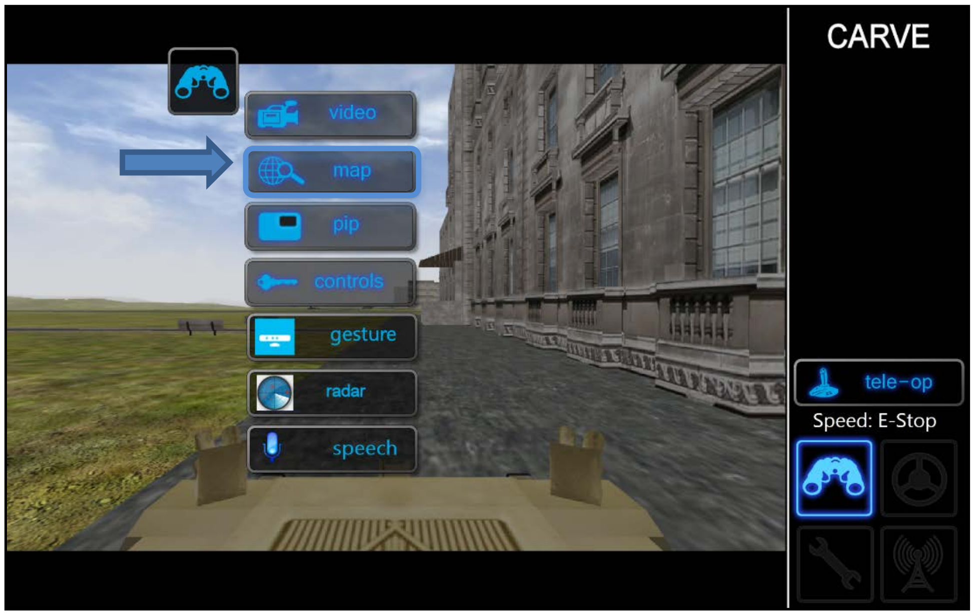


Fig. K-12 Saving the map in CARVE: open the map

9. **Select Map Save Icon.** Once the map is opened, choose the select map save icon (Fig. K-13).

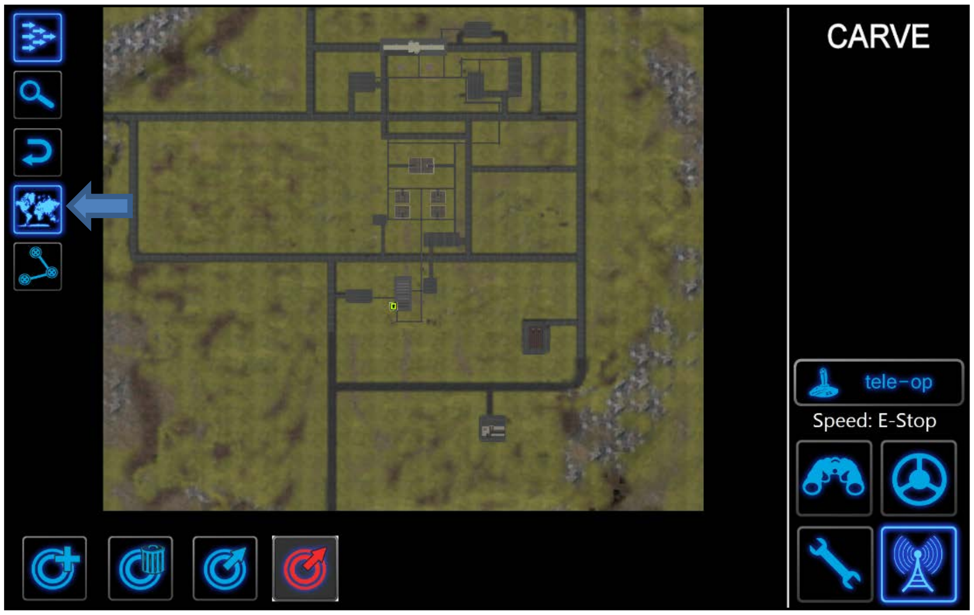
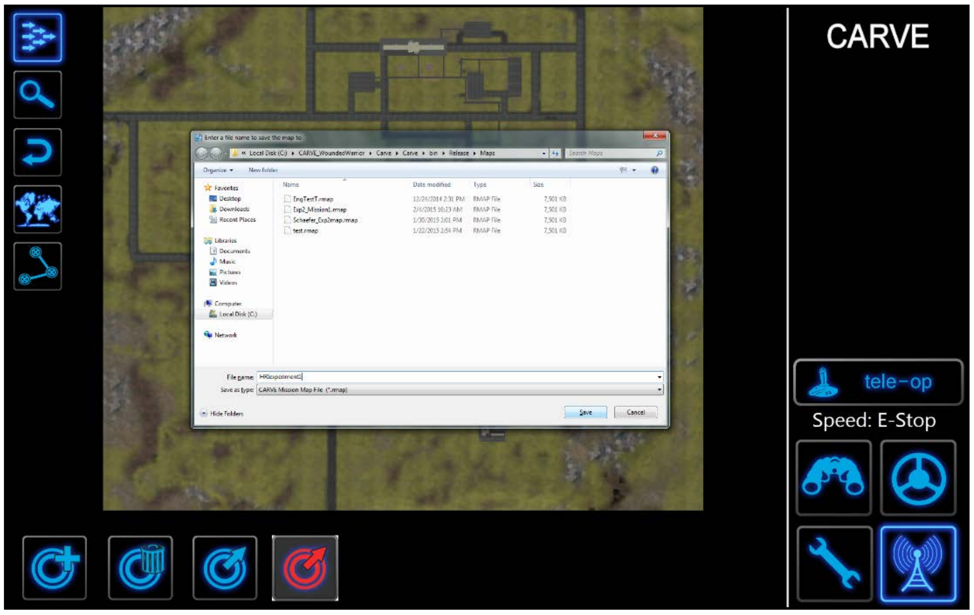


Fig. K-13 Saving the map in CARVE: select map save icon

10. **Save Map:** When saving the map, use a unique Map name and write it down because it is used when creating a CARVE Mission-specific shortcut (Fig. K-14).



Name	Date modified	Type	Size
EngTestT.rmap	12/24/2014 2:31 PM	RMAP File	7,501 KB
Exp2_Mission1.rmap	2/4/2015 10:23 AM	RMAP File	7,501 KB
Schaefer_Exp2map.rmap	1/30/2015 2:01 PM	RMAP File	7,501 KB
test.rmap	1/22/2015 2:54 PM	RMAP File	7,501 KB

File name: HRIexperiment1

Save as type: CARVE Mission Map File (*.rmap)

Fig. K-14 Saving the map in CARVE: assign a unique map name

11. **Close CARVE.** Exit the program by selecting the exit program icon (Fig. K-15).

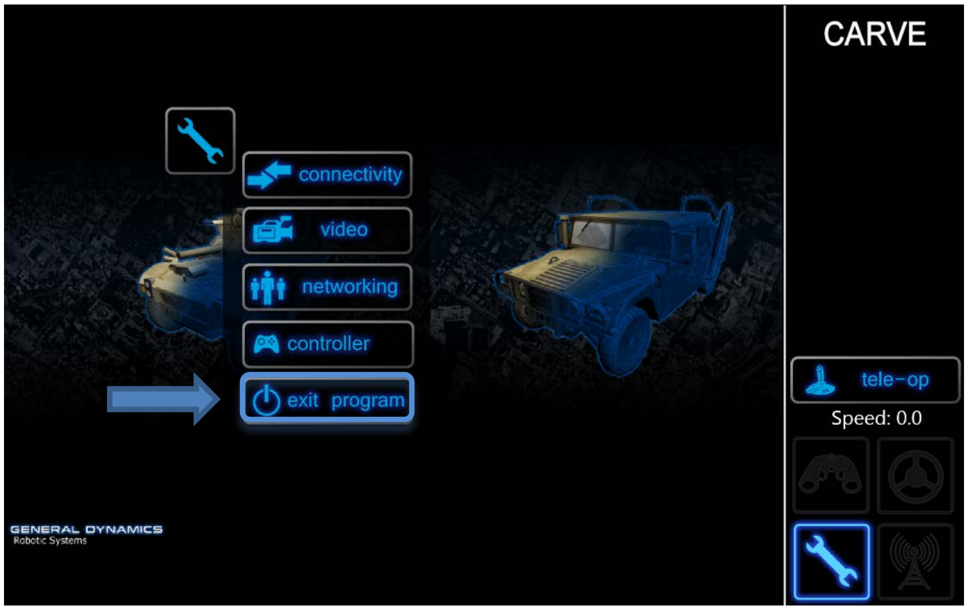


Fig. K-15 Saving the map in CARVE: exit the program

12. **Create CARVE Mission Shortcut.** This shortcut pre-loads the map and reduces wait time (Fig. K-16). On the desktop, right click on the CARVE shortcut, and select Create Shortcut. Rename the new CARVE shortcut.

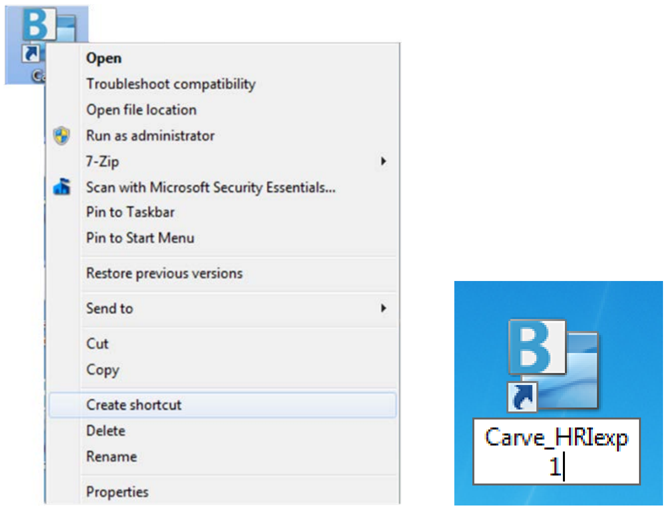
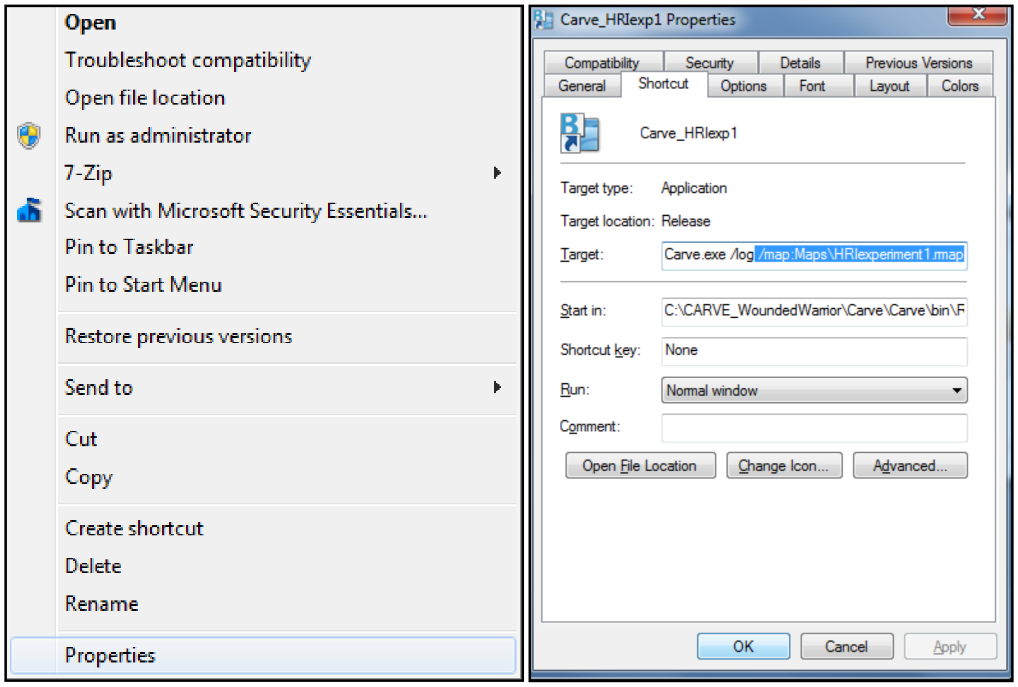


Fig. K-16 Create a CARVE shortcut for a specific mission

13. **Edit Properties.** Right click on the new shortcut, and select properties. Add a space and /map:Maps\HRIexperiment1.rmap (or whatever you named your map) to the end of the Target (Fig. K-17). Now, the map will instantaneously load when selecting the new CARVE shortcut.



Carve_HRIexp1

Target type: Application

Target location: Release

Target: Carve.exe /log /map:Maps\HRIexperiment1.rmap

Start in: C:\CARVE_WoundedWarrior\Carve\Carve\bin\F

Shortcut key: None

Run: Normal window

Comment:

Fig. K-17 Naming the new CARVE shortcut with the specific mission map

Appendix L. CARVE Directions for Waypoint Following

1. **Adding Waypoints for Semi-Autonomous Navigation.** Open a User Mission within CARVE (see Appendix J, steps 1–6). Open the Map (see Appendix J, step 8). The following waypoint options are available (Fig. L-1):

   A. Add waypoint
   B. Delete waypoint
   C. Move waypoint
   D. Move spot report point
   E. Map position (on/off). On: Center vehicle on map; Off: Click on vehicle and move around map
   F. Zoom
   G. Return zoom to 0%
   H. Save map
   I. Save waypoints
   J. Slider for zooming map

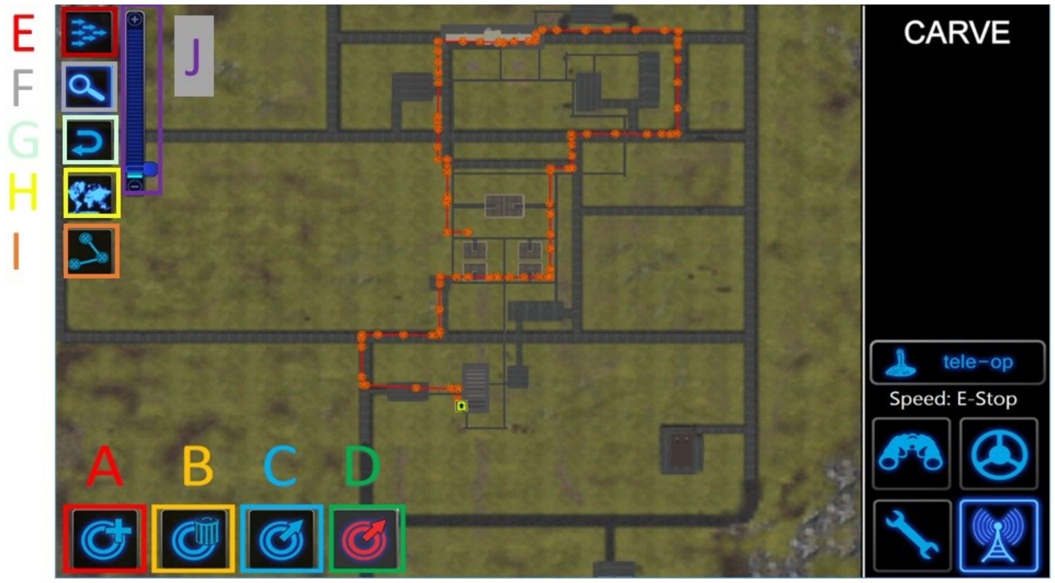


Fig. L-1 CARVE waypoint options

2. **Add Waypoints.** Click on the bull's-eye with the plus sign to add waypoints (Fig. L-2). Then click in the map. The first waypoint should be close to where the vehicle is located. The mouse is now an orange bull's-eye.

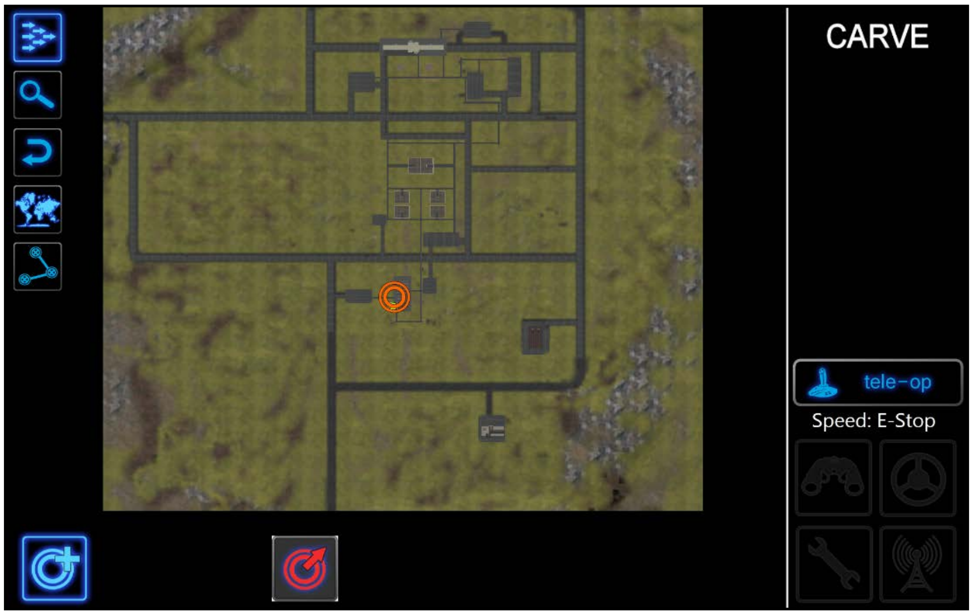
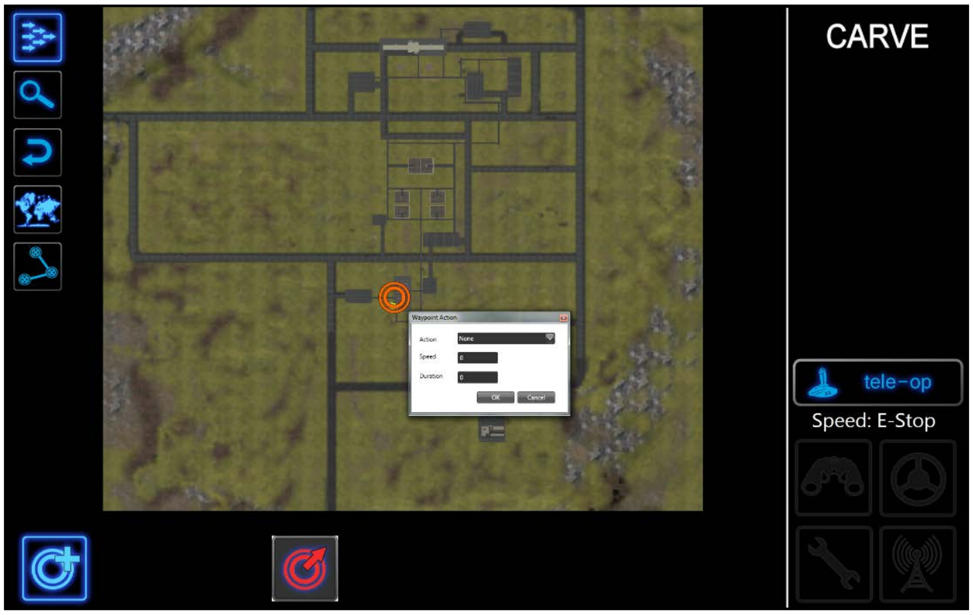


Fig. L-2 How to add navigation waypoints

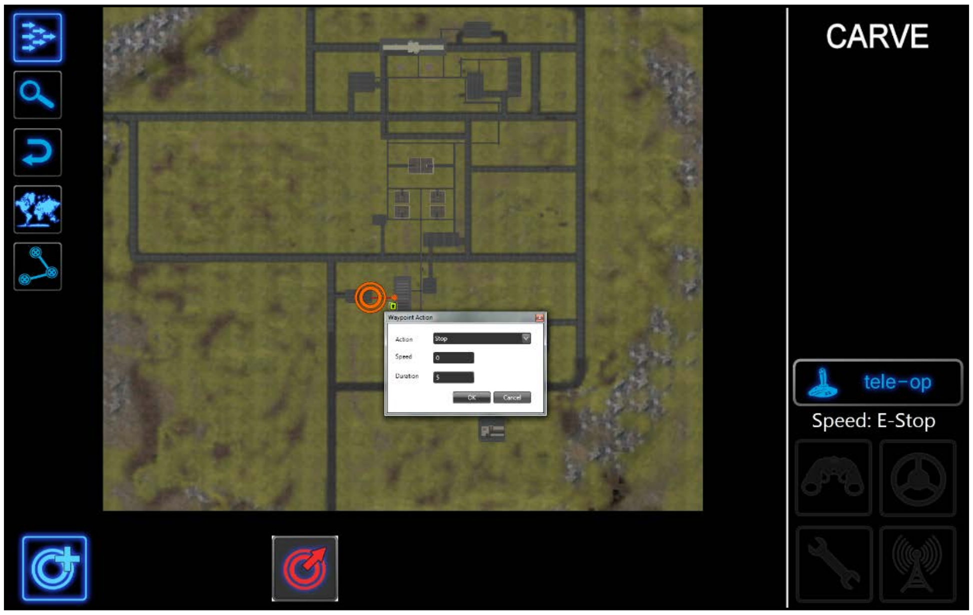
3. **Dynamic Actions.** Every time a waypoint is added, an Action screen will appear (Fig. L-3). This allows the user to customize the waypoints.



Action None
Speed 0
Duration 0

Fig. L-3 How to customize dynamic waypoint actions

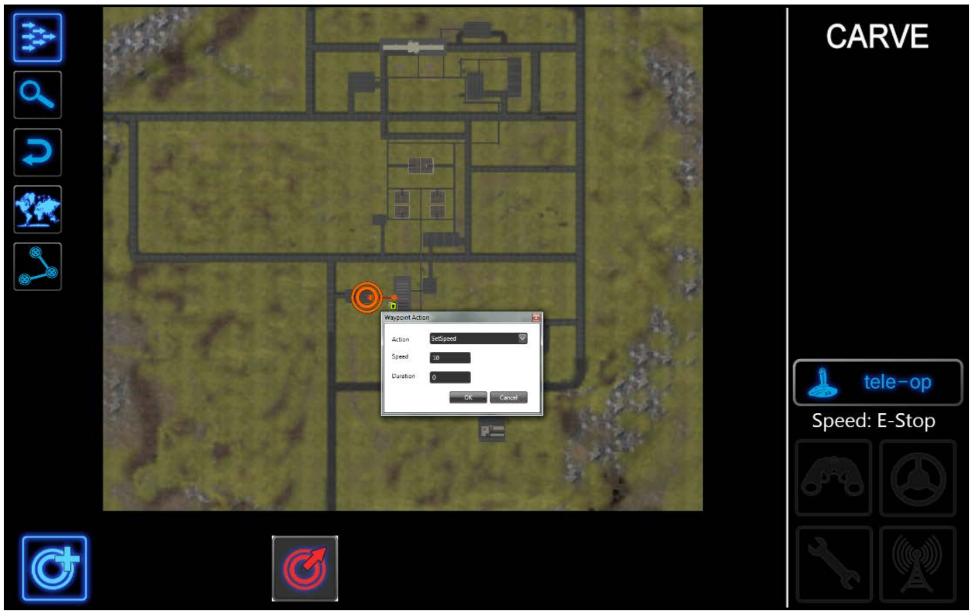
4. **Stop Waypoint.** The Stop waypoint stops the vehicle or robot in that location for a set duration (in seconds) (Fig. L-4).



Action: Stop
Speed: 0
Duration: 5

Fig. L-4 How to use the Stop waypoint action

5. **SetSpeed Waypoint.** This waypoint action allows you to change the speed of the vehicle or robot at a given location (in miles per hour) (Fig. L-5).



Action: SetSpeed
Speed: 10
Duration: 0

Fig. L-5 How to use the SetSpeed waypoint action

6. **Finished Adding Waypoints.** When you have finished adding waypoints to the map, click on the bull's-eye with the plus sign to exit this mode (Fig. L-6).

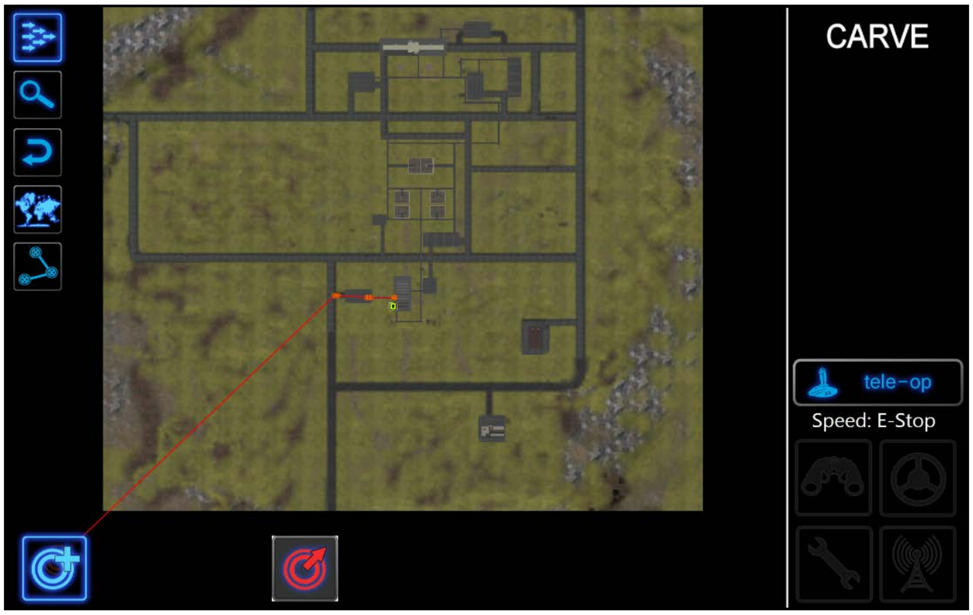


Fig. L-6 How to stop adding waypoints

7. **Adjust Waypoints.** It is possible to adjust the waypoints that were previously placed on the map by clicking on the left mouse button and dragging the waypoint (Fig. L-7).

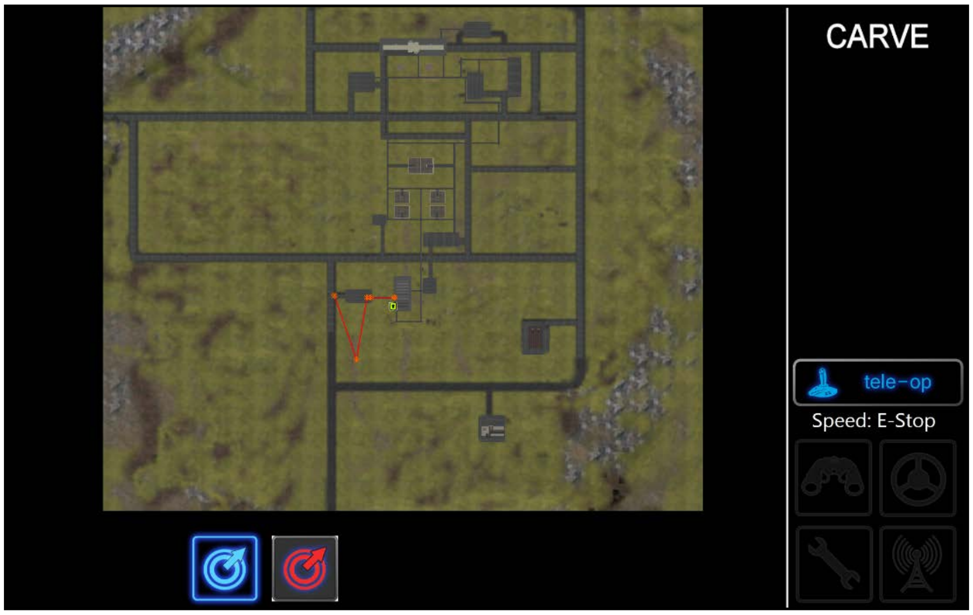
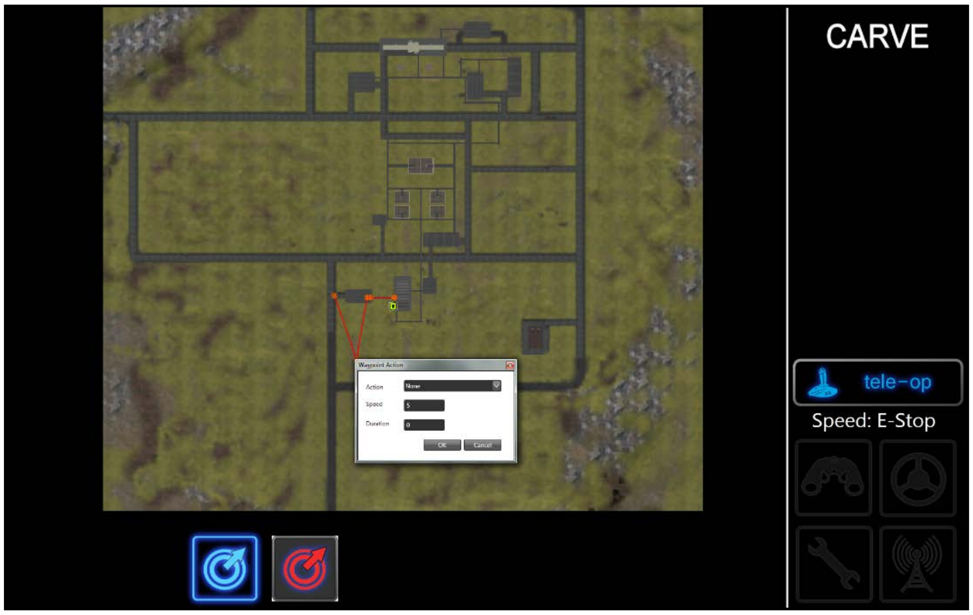


Fig. L-7 How to adjust waypoints

8. **Adjust Actions.** It is also possible to adjust or edit the Actions by clicking on waypoint with the right mouse button (Fig. L-8).



Waypoint Action

Action: None

Speed: 5

Duration: 0

Fig. L-8 How to adjust waypoint actions

9. **Close Edit Menu.** To close this menu, select the blue box (Fig. L-9).

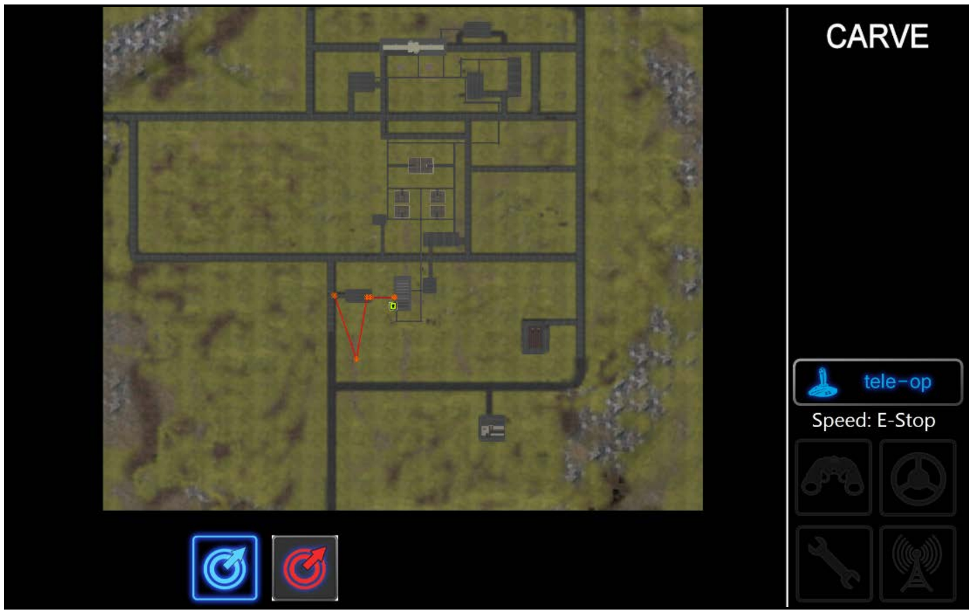
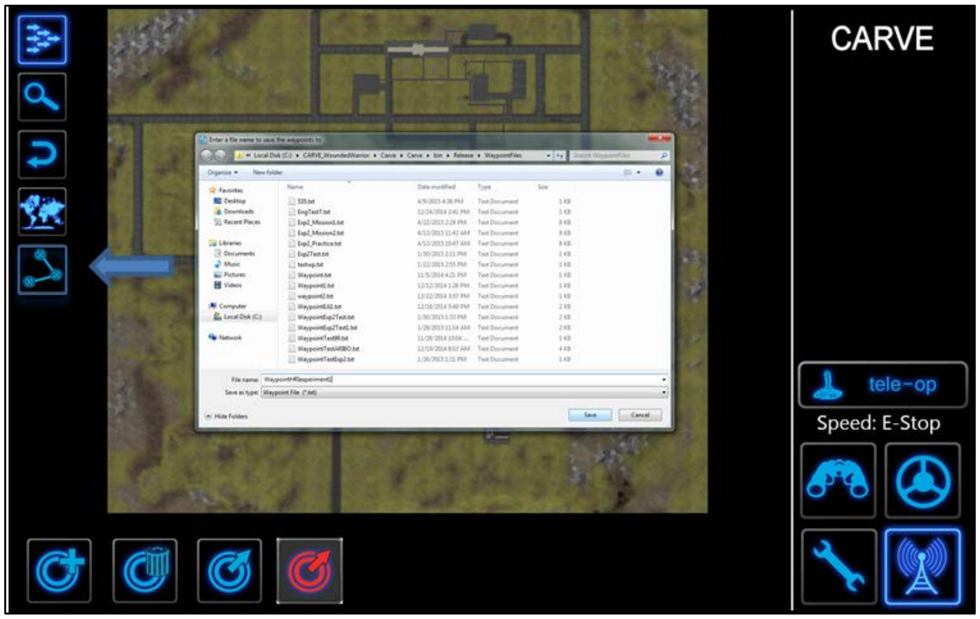


Fig. L-9 How to close waypoint edit menu

10. **Save Waypoints.** Select the "save waypoints" button on left-hand side of the screen (Fig. L-10). This will open a save window. Enter a unique file name (e.g., WaypointsHRIexperiment1) and click Save. Write down the file name because it will be needed for step 12.



Name	Date modified	Type	Size
535.txt	4/9/2015 4:38 PM	Text Document	1 KB
EngTestT.txt	12/26/2014 2:41 PM	Text Document	1 KB
Exp2_Mission1.txt	4/22/2015 2:24 PM	Text Document	8 KB
Exp2_Mission2.txt	4/13/2015 11:42 AM	Text Document	8 KB
Exp2_Practice.txt	4/13/2015 10:47 AM	Text Document	8 KB
Exp2Test.txt	1/30/2015 2:11 PM	Text Document	1 KB
testwp.txt	1/22/2015 2:55 PM	Text Document	1 KB
Waypoint.txt	11/5/2014 4:21 PM	Text Document	1 KB
Waypoint1.txt	12/12/2014 1:28 PM	Text Document	1 KB
waypoint2.txt	12/22/2014 3:57 PM	Text Document	1 KB
WaypointEA1.txt	12/16/2014 5:49 PM	Text Document	2 KB
WaypointExp2Test.txt	1/30/2015 1:33 PM	Text Document	2 KB
WaypointExp2Test1.txt	1/28/2015 11:04 AM	Text Document	2 KB
WaypointTest99.txt	11/26/2014 10:04 ...	Text Document	1 KB
WaypointTestARIBO.txt	12/10/2014 9:02 AM	Text Document	4 KB
WaypointTestExp2.txt	1/26/2015 1:31 PM	Text Document	1 KB

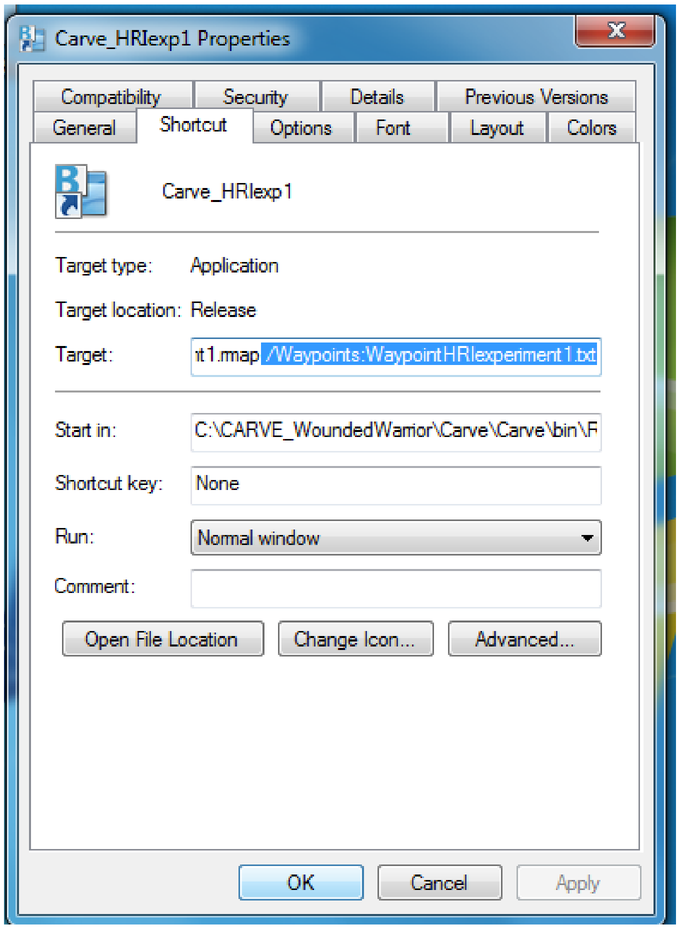
File name: WaypointHRIexperiment1

Save as type: Waypoint File (*.txt)

Fig. L-10 How to save waypoints

11. **Exit CARVE.** Exit CARVE to set up the application to automatically load the waypoint file in the future.

12. **Setting up Automatic Loading of Waypoints.** Right click on the CARVE shortcut, click on properties, and add a space, followed by /Waypoints:WaypointHRIexperiment1.txt in the Target (Fig. L-11). Press Apply and Ok. Now when you log back into CARVE, your saved map and waypoints will already be loaded. Waypoints can be edited at any time, just remember to save.


Carve_HRIexp1 Properties

Carve_HRIexp1

Target type: Application

Target location: Release

Target: it1.map /Waypoints:WaypointHRIexperiment1.txt

Start in: C:\CARVE_WoundedWarrior\Carve\Carve\bin\R

Shortcut key: None

Run: Normal window

Comment:

Fig. L-11 Set up automatic loading of waypoints in the CARVE shortcut properties

Appendix M. Directions to Set Up the Radar Task

1. **Setting up the Radar Task.** CARVE is set up to allow users to accomplish a secondary task while attempting to conduct a primary task such as monitoring the unmanned vehicle. This appendix provides the steps needed to set up and run the radar task. Once the CARVE application is open, click on the Steering Wheel (a User Mission from RIVET is not required during setup of the radar task) (Fig. M-1).

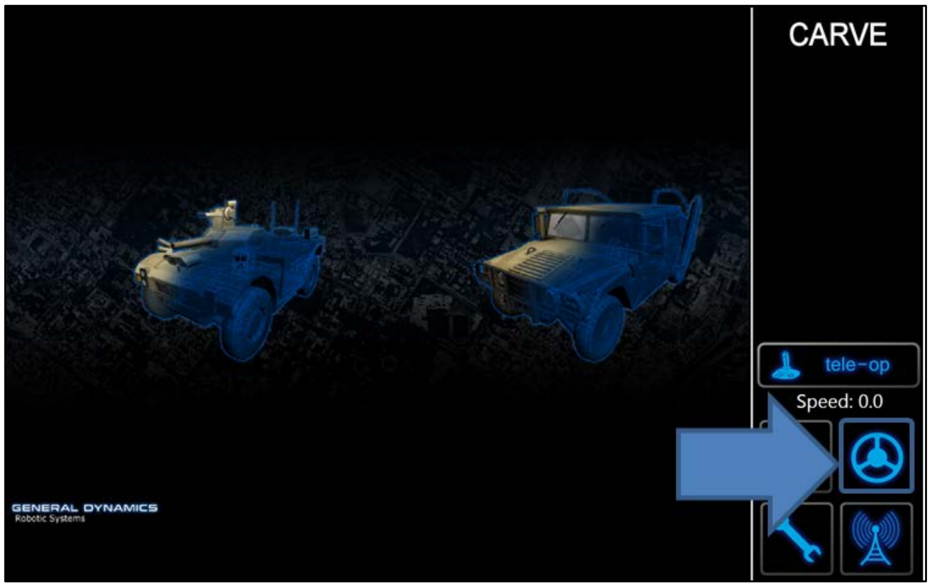


Fig. M-1 Select the Vehicle Operations submenu to set up the task

2. **Open Radar Menu.** To open the Radar menu, click on Waypoint Follow and Save Waypoints icons (Fig. M-2).

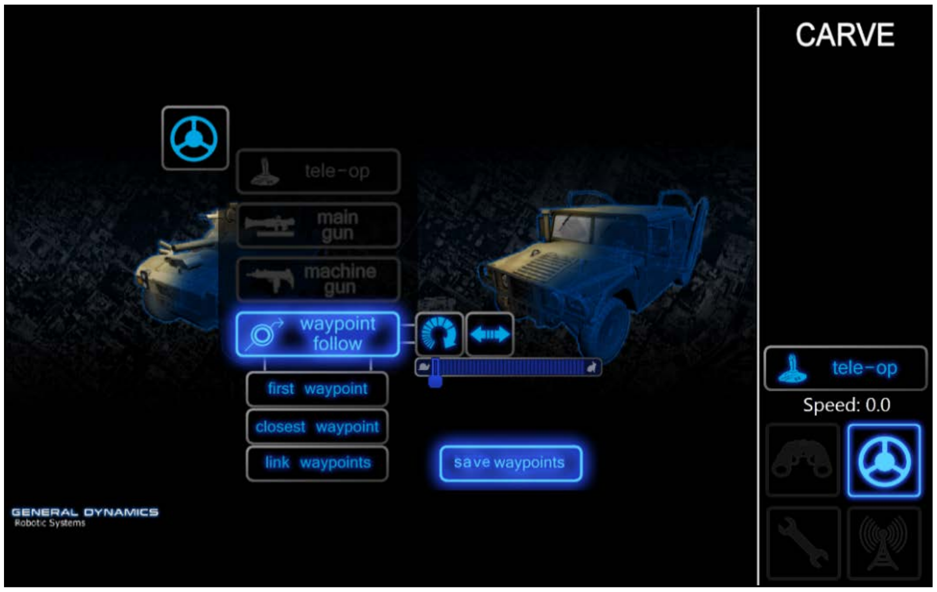


Fig. M-2 Open radar menu

3. **Radar Run Setup Screen.** The Run Setup Screen GUI allows the researcher to prepare the user for the upcoming data collection. As shown in Fig. M-3, the run name (1) is entered along with the path name (2) if a secondary task will be used during the run. The radar check box (4) must be selected for the radar task to start. The safety zone (5) sets the radius from the center of the radar screen. Any blip that enters this safety zone is then changed from friendly to enemy by changing the color from yellow to red. The path name coincides with a text file that is created by clicking on the create path file (3).

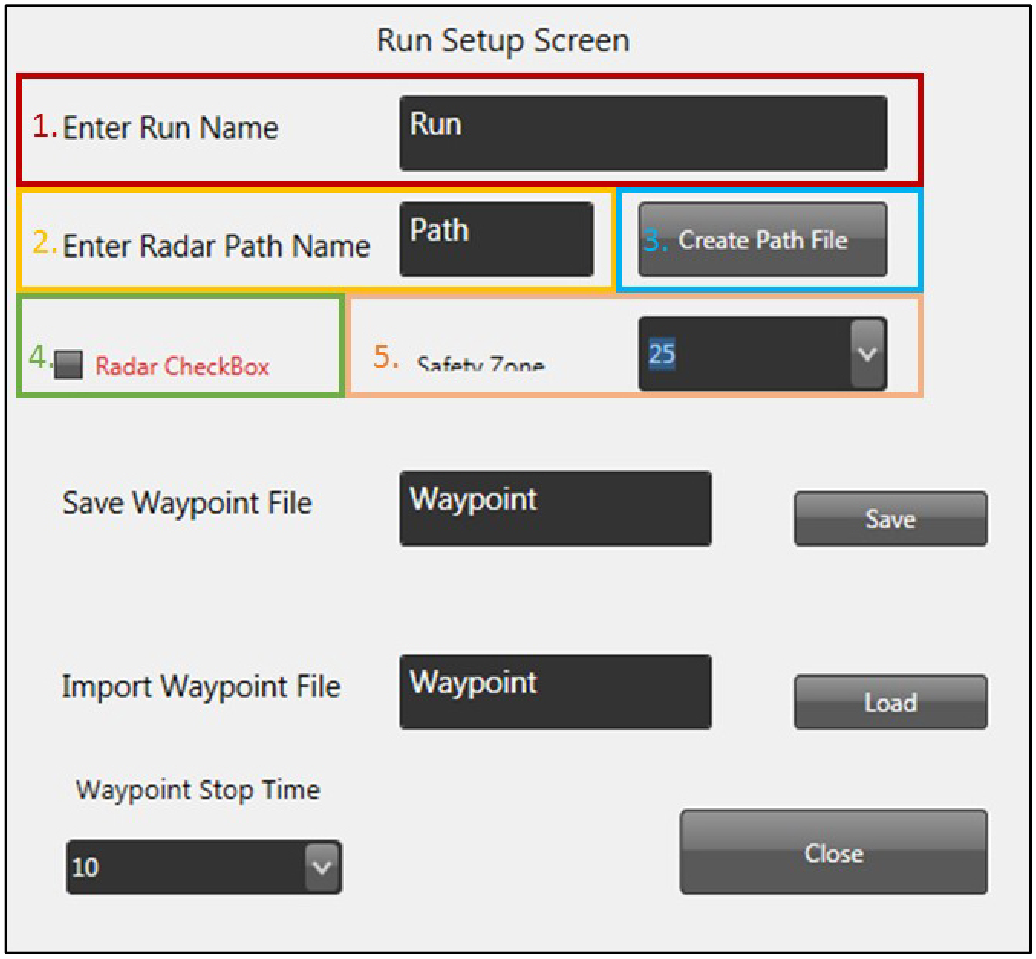
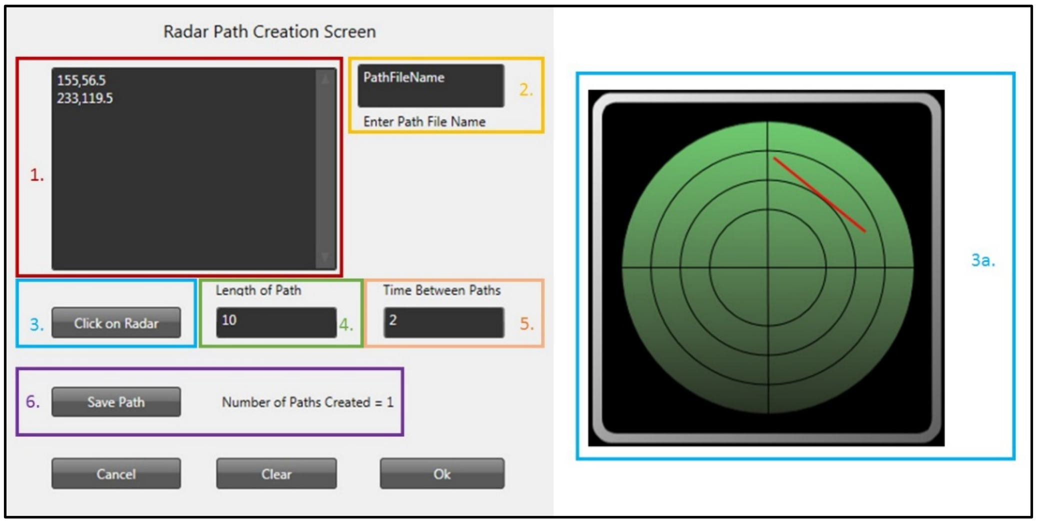


Fig. M-3 Radar Run Setup Screen GUI

4. **Radar Path Creation.** The button opens the Radar Path Creation Screen (Fig. M-4). The X and Y coordinates are shown in the list box (1). The text file name is entered in the text box (2). To enter the point, press the Click on Radar point button (3) and then click inside the radar screen (3a). This will enter the point in (1). This process is repeated to make a path. Upon the addition of a second and any successive points, the path is drawn in red inside the radar screen (3a). Once the path is complete, enter a path length in seconds (4) as well as the length of pause between this path and the next one (5). If satisfied with the path, click save path (6) and the number of completed paths will increase in the label to the right of the save button. Once all the paths are completed, click Ok. Clicking the Clear button will clear all the data from the screen, while the cancel button will close out the GUI without saving anything.



155,56.5
233,119.5

PathFileName

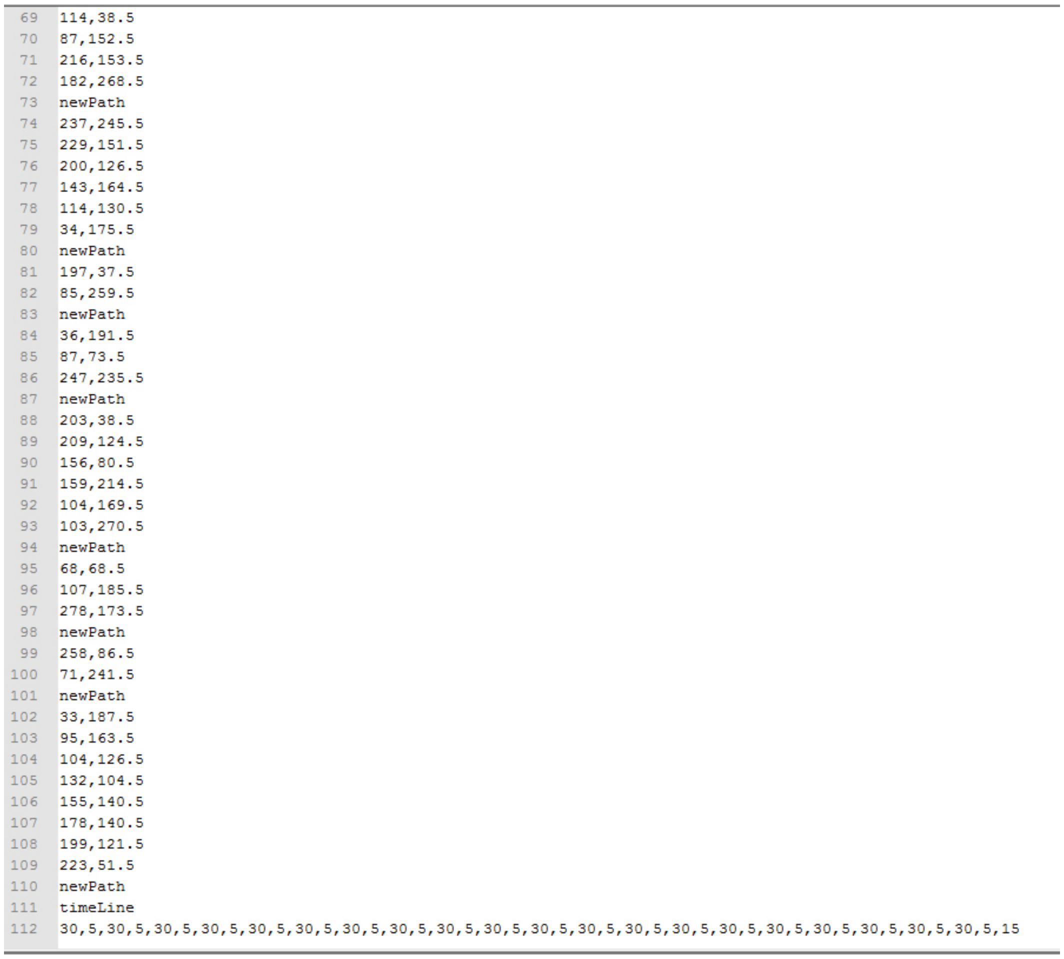
Length of Path: 10

Time Between Paths: 2

Number of Paths Created = 1

Fig. M-4 Radar Path Creation Screen GUI

5. **Path Data Output.** The default file name entered is PathFileName and should be changed by the user. It is recommended that radar files be named according to run name to help with setup and record keeping. This file is saved in the bin\Release\Radar folder. Each file should have a unique name. If the path file name is the same as a previous file, it will overwrite that previous file. The radar path text file has X and Y coordinates on the same line separated by a comma. For parsing, each path is separated by newPath (Fig. M-5). The timeline is followed by a series of numbers that represent the time to complete the path followed by the time between paths.



```
114,38.5
87,152.5
216,153.5
182,268.5
newPath
237,245.5
229,151.5
200,126.5
143,164.5
114,130.5
34,175.5
newPath
197,37.5
85,259.5
newPath
36,191.5
87,73.5
247,235.5
newPath
203,38.5
209,124.5
156,80.5
159,214.5
104,169.5
103,270.5
newPath
68,68.5
107,185.5
278,173.5
newPath
258,86.5
71,241.5
newPath
33,187.5
95,163.5
104,126.5
132,104.5
155,140.5
178,140.5
199,121.5
223,51.5
newPath
timeLine
30,5,30,5,30,5,30,5,30,5,30,5,30,5,30,5,30,5,30,5,30,5,30,5,30,5,30,5,30,5,30,5,30,5,30,5,30,5,30,5,15
```

Fig. M-5 Sample data output of radar paths

INTENTIONALLY LEFT BLANK.

List of Symbols, Abbreviations, and Acronyms

3-D	3-dimensional
AAR	after-action review
ARIBO	applied robotics for installations and base operations
ARL	US Army Research Laboratory
BOLT	basic operations layer transmissions
CACTF	combined arms collective training facility
CARVE	Control of Autonomous Robotic Vehicle Experiment
GDRS	General Dynamics Robotic Systems
GUI	graphical user interface
HITL	human-in-the-loop
HRI	human-robot interaction
IP	Internet protocol
LAN	local area network
NPC	non-player character
PIP	picture-in-picture
RCTA	Robotics Collaborative Technology Alliance
RIVET	Robotic Interactive Visualization and Experimentation Technology
SQL	structured query language
TAC-C	tactical autonomous combat chassis
TF	task force
TGE	Torque Game Engine
UAV	unmanned air vehicle
UDP	user datagram protocol

VE	virtual environment
WEI	World Editor Inspector

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL AND RECORDS MGMT
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
14 (PDF)	DIR USARL RDRL HR L ALLENDER P FRANASZCZUK RDRL HRM K MCDOWELL RDRL HRS J LOCKETT K OIE RDRL HRS E S HILL K SCHAEFER RDRL VTA J BORNESTEIN M FIELDS R BREWER M CHILDERS C LENNON H EDGE C KRONIGER
3 (PDF)	ROBOTICS CLLBRTV TECHNLGY ALLNCE GEN DYNAMICS D PATEL E WELLER C DIBERARDINO
4 (PDF)	UNIV OF CENTRAL FLORIDA F JENTSCH P HANCOCK S FIORE D BARBER

INTENTIONALLY LEFT BLANK.